

COMMODORE 64, DESDE EL COMIENZO



AULA DE INFORMATICA APLICADA

Commodore-64. Desde el comienzo

Aula de Informática Aplicada

Comité de redacción:
Manuel Garrido Pérez
Prof. Titular de la Facultad de Informática, U.P.M.
M. Teresa Gómez-Mascaraque Pérez
Licenciada en Ciencias Químicas

Primera edición, 1986

© EDITORIAL ALHAMBRA, S.A., para la presente edición
R.E. 182
28001 Madrid. Claudio Coello, 76

Delegaciones:
08008 Barcelona. Enrique Granados, 61
48014 Bilbao. Iruña, 12
18009 Granada. Pza. de las Descalzas, 2
15005 La Coruña. Pasadizo de Pernas, 13
28002 Madrid. Saturnino Calleja, 1
33006 Oviedo. Avda. del Cristo, 9
38004 Santa Cruz de Tenerife. General Porlier, 14
41012 Sevilla. Reina Mercedes, 35
46003 Valencia. Cabillers, 5
47014 Valladolid. Gavilla, 3
50005 Zaragoza. Concepción Arenal, 25

México
Editorial Alhambra Mexicana, S.A.
Avda. División del Norte, 2412
03340 México, D.F.

Argentina
EDICLE
Juncal, 4649/51
1425 Buenos Aires

Colombia
Editorial Alhambra Colombiana, Ltda.
Calle 117 n° 11, A-65
Bogotá

nc 06120100

ISBN 84-205-1327-X

Depósito legal: M. 9.513.—1986

Cubierta: Estudio Enlace, S.A.
Fotocomposición: Estudio Enlace, S.A.
Impresión: A. G. Benzal, S.A.
Papel: Kanguros
Encuadernación: Gómez Pinto, S.A.

Indice

Prólogo

Durante muchos años la informática ha sido una actividad reservada a profesionales técnicamente muy cualificados, a los que muchas veces se consideraba «semimagos» por ser capaces de «convencer» a una máquina para que realizase tareas que hasta entonces se había pensado sólo podrían ser resueltas por el hombre. La sorpresa era aún mayor: cálculos que a un hombre le hubieran llevado gran parte de su vida concluir eran efectuados en apenas unas horas por los «cerebros electrónicos».

Ordenadores como su Commodore 64 han colaborado intensamente a acabar con este mito al ofrecer, junto con una gran capacidad, un precio muy asequible que pone a la mano de cualquiera la posibilidad de dar el «pequeño gran salto» sin más que interesarse por el tema.

Hemos escrito este libro para ayudarles «desde el principio», suponiendo que quizá sea su primer contacto con el ordenador.

El primer capítulo le será de gran utilidad para conocer cómo hacer funcionar a su máquina. Encontrará sentido a cada conexión y sobre todo podrá *familiarizarse* con el teclado, herramienta indispensable para comunicarse con éxito con el ordenador.

El principal objetivo del libro es que domine el lenguaje BASIC con el que su Commodore 64 viene equipado. Cada nueva instrucción será menos nueva para usted, porque según avance irá sintiendo la necesidad de poder hacer algo más. Así, cada vez que le presentemos otra *palabra* BASIC sentirá que ya intuía su existencia.

Cada uno de los programas que le presentamos le aportará alguna idea nueva que le servirá para resolver los problemas que tiene en su mente. Intente hacerlos por su cuenta, no se limite a teclearlos porque, aunque le parezca que lo va entendiendo todo con facilidad, el gran paso lo dará en el momento en que diseñe el primer programa por su cuenta. Le llenará de satisfacción y le animará a seguir. Consiéntanos un consejo final que posiblemente haya recibido otras veces, pero que no por ello resulta intrascendente: No se quede nunca con la duda «qué ocurriría si hiciese...» Hágalo, no le importe equivocarse, aprenderá de sus errores. Tenga presente que, a diferencia de lo que pueda ocurrir con otras materias, a programar se aprende programando y no simplemente leyendo.

1 Introducción

Probablemente, cuando abra este libro ya habrá desempaquetado y comenzado a instalar su Commodore 64. No se precipite. Lo primero que hemos de hacer es buscar un escondite de trabajo cómodo y bien iluminado. Lo ideal es una mesa plana de unos 70/75 cm de alto por 70/80 cm de fondo y tan ancho como le sea posible. Pronto empezará a acumular periféricos, libros, apuntes, cassettes, disquetes, etc., y no le parecerá exagerada esta precaución.

DESEMBALADO

Coloque el contenido del paquete encima de la mesa y empiece a examinarlo. Tenemos:

1. El ordenador C-64.
2. Una fuente de alimentación.
3. Un cable para conectar el C-64 con el televisor.



Fig. 1.1. Equipo básico del Commodore 64.

También habrá adquirido un dispositivo de almacenamiento, bien la Datassette o bien la unidad de disquetes y, si se ha sentido espléndido en su inversión inicial, puede que además disponga de una impresora y de un modem. A lo largo de este libro supondremos que la configuración de que dispone está formada por la configuración básica y la Datassette. Si acaso dispone de otros periféricos, acuda al apéndice «Otros periféricos», que se ocupa de exponerle en forma detallada su utilización.

Tras desempaquetar el Commodore, lo que más salta a la vista es lo variopinto de su teclado, lleno de letras, dígitos, caracteres especiales, abreviaturas de palabras, palabras completas, etc. Pero, como lo primero que necesitamos es ponerlo en funcionamiento, vamos a examinar detenidamente su sistema de conexiones. Estas están situadas en el lateral derecho y en la parte posterior del ordenador (véanse Figs. 1.2 y 1.3)

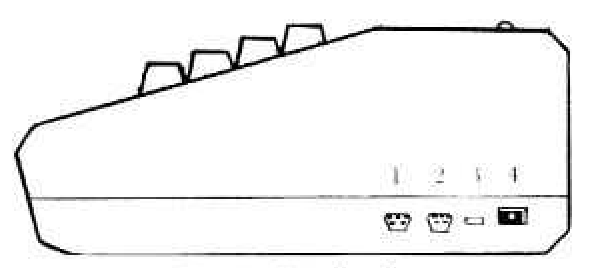


Fig. 1.2. Vista lateral

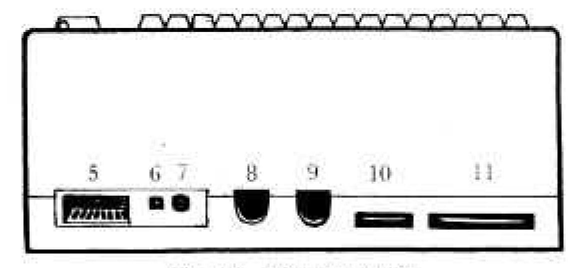


Fig. 1.3. Vista posterior

1. Conexión para Juegos 1 (Game Port 1).
2. Conexión para Juegos 2 (Game Port 2).
3. Interruptor de alimentación.
4. Conexión de alimentación.
5. Conexión para cartuchos.
6. Selector de canal.
7. Conexión para televisión.
8. Conexión audio/video (Video Port).
9. Conexión serie (Serial Port).
10. Conexión para Datassette.
11. Conexión paralelo de usuario (Parallel User Port).

CONEXIONES

Vamos a hablar en primer lugar de las conexiones 3, 4, 7 y 10, que son las que necesitamos para nuestra configuración básica (suponiendo que ha decidido utilizar su televisión).

3. *Interruptor de alimentación.* ON/OFF (conexión/desconexión). Una vez conectado el sistema, utilizaremos este interruptor para encender y apagar la

unidad central. Recuerde siempre que al apagar la unidad central perderá definitivamente todo lo que haya estado trabajando, a menos que lo haya almacenado en la Datassette o en la unidad de discos.

Recuerde también que al desconectar la unidad central la fuente de alimentación seguirá funcionando, a menos que desconecte el enchufe de la red. Es importante recordar esto, pues frecuentemente el dispositivo que más problemas ocasiona es la fuente de alimentación. ¡Cúidela!

4. *Conexión de alimentación.* La fuente de alimentación tiene dos cables: uno termina en un enchufe normal para conectarlo a la red y el otro termina en una clavija típica de siete pins, pero en este caso sólo tiene cuatro. No se preocupe, no falta ninguna, viene así. Para desconectar el ordenador, lo adecuado es utilizar el interruptor ON/OFF, del que hablamos en el punto anterior. No se acostumbre a sacar y meter la clavija de conexión de alimentación, pues se acabará deteriorando, tomando holgura, y esto traerá consigo fallos intermitentes en la alimentación, que arruinarán sus ganas de trabajar con el ordenador.

7. *Conexión para la televisión.* En el paquete de su Commodore viene un cable de antena para conectarlo con su aparato de televisión. Si la TV es antigua puede que necesite un adaptador para conectar con el enchufe de UHF. En la ferretería más próxima a su casa encontrará el material necesario, y es muy sencillo de construir y barato. En caso de que no necesite adaptador, conecte el enchufe (del cable de antena) que contiene la clavija más larga a su Commodore y el otro al televisor.

10. *Conexión para Datassette.* La Datassette viene provista de un solo enchufe, con forma plana, que va conectado al Commodore. No se preocupe porque no tenga otro enchufe para conectarlo a la red, su Commodore se encargará

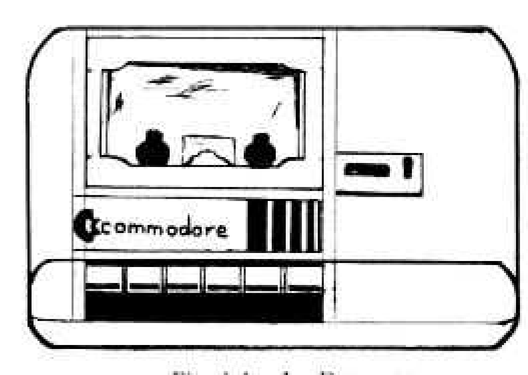


Fig. 1.4. La Datassette.

de suministrarle la energía para que funcione. Observe el enchufe de la Datassette y verá una pequeña lámina de plástico que divide la conexión en dos zonas. Al enchufar la Datassette a su ordenador fíjese en que coincida esta lámina de plastico con la ranura que presenta la tarjeta de conexión del Commodore.

Quedan por mencionar diversas conexiones. Sin embargo, si su configuración es la básica (ordenador, TV, Datassette) puede encontrar más útil saltar, de momento, el resto de este apartado y pasar al siguiente. Siempre tendrá tiempo para volver a leerlo.

1 y 2. *Conexiones para juegos.* Aquí es donde tendrá que conectar los diversos controladores de juegos, tales como los *joysticks* y los *paddles,* así como los lápices ópticos y otros dispositivos especiales. Además, algunos programas, como medida de seguridad, se venden con una pequeña caja negra, llamada «llave», que es preciso conectar a uno de estos enchufes para que el programa funcione.

5. *Conexión para cartuchos,* Poco a poco irá encontrando múltiples usos para esta conexión. Además de servir para utilizar los programas y juegos que vienen en cartuchos, existen adaptadores que nos permiten utilizar unidades de disquetes distintas de las de Commodore y muchas cosas más.

6. *Selector de canal.* No se preocupe, en general no tendrá que manipular este dispositivo. Su finalidad es la de ser utilizado en caso de que no pueda variar la sintonización de los canales de su televisor. En este caso tendrá que variar el canal de emisión de su Commodore (mediante el tornillo que ve en la ranura) hasta que éste coincida con algún canal de su televisor. Si, como es habitual, puede variar la sintonización de los canales de su televisor, no toque este dispositivo.

8. *Conexión audio/vídeo.* Si tiene un monitor monocromático o uno de color es aquí donde hay que conectarlo. En este conector puede introducir también un enchufe para enviar el sonido a un amplificador y, si es hábil con la electrónica, puede construir un cable que a partir de esta toma envíe la imagen a un monitor y el sonido a un amplificador. A través de esta conexión el Commodore puede también recibir sonido de una fuente externa, tal como un cassette, un micrófono etc.

9. *Conexión en serie (Serial Port).* Aquí es donde ha de enchufar el dispositivo de disquetes 1541, la impresora 1525 y cualquier otro dispositivo que utilice una configuración de entrada/salida en serie. En el apendice «Otros periféricos» encontrará las instrucciones para conectar la unidad de disquetes y la impresora.

11. *Conexión de usuario.* También llamada conexión en paralelo. Aquí es donde el usuario podrá aplicar la mayoría de sus ideas. En principio podemos conectar el modem, pero si usted es experimentador encontrará que ésta es la conexión a utilizar para sus experiencias caseras.

INSTALACION

Bien, vamos a poner en funcionamiento nuestro Commodore. Una vez elegida la mesa y una silla confortable (tras varias horas de trabajo, su espalda agradecerá esta recomendación), empecemos por colocar el televisor. Coloque después el Commodore y la Datassette y comience con las conexiones. En primer lugar conecte el cable que va de la fuente de alimentación al C-64 (no enchufe todavía el cable a la red), después introduzca la conexión de la Datassette en el C-64 y coloque el cable de antena que va desde el C-64 al televisor. Ya hemos realizado todas las *conexiones internas*. Si al intentar alguna conexión observa cierta resistencia no trate de introducirlo a la fuerza, sólo conseguirá estropear la conexión. Compruebe en las figuras 1.2 y 1.3 si está utilizando el conector adecuado y, si es así, mire si lo está haciendo correctamente. En todo caso, en lugar de empujar con fuerza, siempre es aconsejable intentarlo varias veces con suavidad.

Ahora sólo quedan por realizar las conexiones a la red (habrá tenido la precaución de que cerca de la mesa hubiera un enchufe, ¿no?). En principio necesitamos realizar sólo dos conexiones: la del televisor y la de la fuente de alimentación del Commodore (el Datassette se alimenta a través del C-64), pero después iremos necesitando enchufes para la unidad de disquetes, la impresora..., así que es aconsejable estar bien provisto de enchufes múltiples.

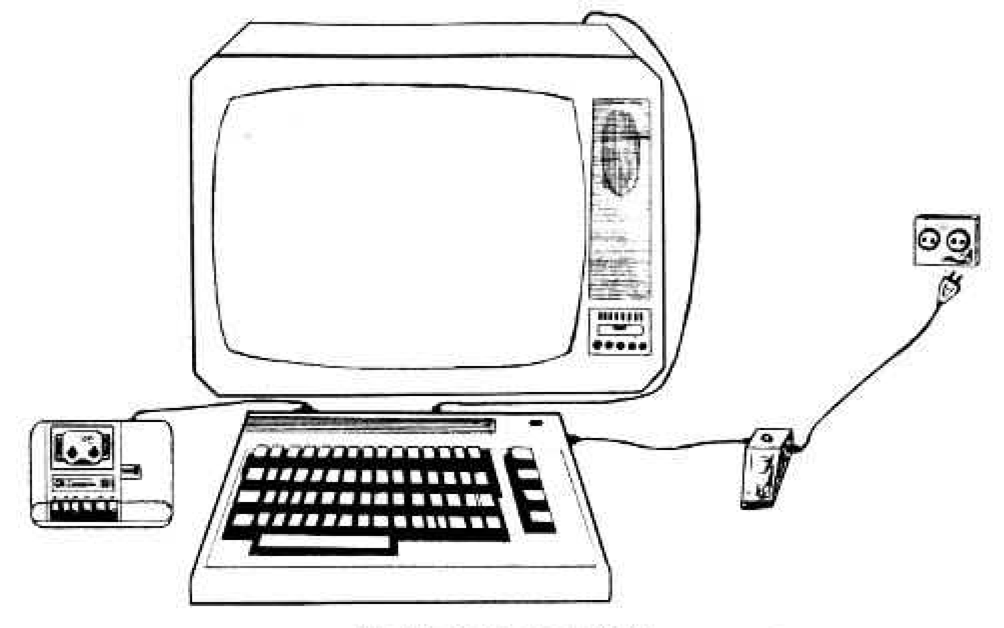


Fig. 1.5. Instalación del C-64.

Una vez realizadas las *conexiones externas* (a la red), pulse el interruptor de alimentación del C-64 y el interruptor del televisor, para encender ambos, y espere unos segundos para ver el mensaje del Commodore en la pantalla. Es difícil que obtenga este mensaje la primera vez que lo intente, puede que tenga suerte y lo único que haya que hacer sea ajustar los controles de brillo y contraste de su televisor, pero lo normal es que tenga que dedicar cierto tiempo a la sintonización de su televisor. ¡Ay qué suerte los que disponen de mandos de sintonización automática! Pero si éste no es el caso, paciencia. Si su televisor es

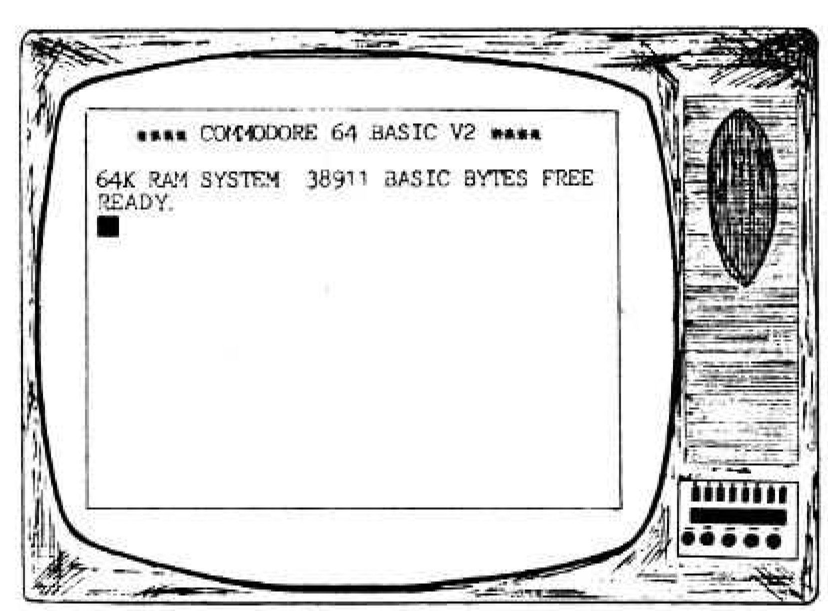


Fig. 1.6. Saludo del Commodore.

sólo de dos canales, VHF y UHF, tendrá que utilizar el canal de UHF y girar la ruedecilla de sintonización hasta visualizar el mensaje. Si su aparato dispone de diversos canales de sintonización, 4, 6, 8, etc, elija uno de los últimos canales (¡no interfiera con el canal de vídeo que utilice su familia!) para dejarlo constantemente dedicado al C-64. Dedique tiempo a la sintonización, pues el cansancio posterior de sus ojos dependerá de ello. Cuando haya logrado una buena sintonización y se encuentre disfrutando de las tonalidades azules en las que Commodore nos transmite sus mensajes, no se resista, teclee alegremente y sin preocupación, experimente, el Commodore no se estropeará por ello.

EL TECLADO

Para comunicar con el ordenador hemos de dominar dos puntos: el manejo del teclado y el lenguaje con el que podemos hablarle a la máquina. Esto es, tenemos que aprender un lenguaje que el ordenador pueda «entender», y utilizar el teclado para escribirlo correctamente; corregir aquello en que nos hayamos equivocado, etc. El lenguaje que viene incorporado con el C-64 es el BASIC, y de él nos ocuparemos a lo largo del libro. Vamos ahora a iniciarnos en el uso del teclado.

Lo primero que observamos es una gran variedad de teclas: unas contienen números y símbolos; otras letras, y caracteres; otras tienen unas palabras en inglés; otras, una abreviatura, etc. ¡No se asuste! No es tan difícil. En esta sección vamos a introducir el manejo básico del teclado, de modo que en cualquier momento le pueda servir de referencia. Sin embargo, algunas actuaciones específicas las veremos cuando surja la necesidad de utilizarlas. En este momento lo importante es practicar, así que anímese y teclee todo lo que se le ocurra con cada apartado.

Cursor

Recibe este nombre el rectangulito azul claro que aparece parpadeando en la pantalla. Cuando conectamos el ordenador, el cursor aparece situado debajo de la R de READY (ver Fig. 1.6). El cursor indica el lugar de la pantalla en que aparecerá el siguiente carácter que tecleemos. Según vamos escribiendo, el cursor se desplaza hacia la derecha. Cuando llega al final de una línea, el cursor salta al principio de la línea siguiente. (En una línea caben 40 caracteres.)

Teclas SHIFT, SHIFT/LOCK, C= y CTRL

La utilización de las teclas SHIFT (hay dos: una a la izquierda y otra a la derecha), C= y CTRL es parecida a la de la tecla de mayúsculas de una máquina de escribir, esto es, pulsando una de ellas, sin pulsar ninguna otra tecla, no tiene efecto alguno. Para conseguir ciertas acciones tendremos que pulsar una de estas teclas y, mientras la mantenemos pulsada, presionar la tecla deseada. Esto lo iremos viendo a medida que introduzcamos las acciones posibles.

Cuando encendemos la máquina y comenzamos a pulsar las teclas de letra, éstas aparecen como mayúsculas en la pantalla. A esto lo llamaremos *«modalidad mayúsculas»*. Si pulsa una sola vez, y al mismo tiempo, las teclas C= y SHIFT, verá que todas las letras que aparecían en la pantalla pasan inmediatamente de mayúsculas a minúsculas. Ahora, al pulsar teclas de letra, éstas aparecerán en

minúsculas. A esto lo llamaremos *«modalidad minúsculas»*. Para volver a pasar de «modalidad minúsculas» a «modalidad mayúsculas», basta con repetir la acción de pulsar a la vez teclas C= y SHIFT. Pulse diversas veces estas dos teclas y entreténgase observando los cambios en su pantalla.

La tecla SHIFT LOCK sirve, al igual que en la máquina de escribir, para mantener la acción de la tecla SHIFT sin necesidad de tener que estar pulsando constantemente esta última.

Teclas CRSR ↑/↓, *CRSR ←/→*,

Estas reciben el nombre de teclas de desplazamiento del cursor, y, como su nombre indica, sirven para situar el cursor en la posición que deseemos.

Pulse la tecla CRSR←/→ y verá que el cursor se desplaza hacia la derecha. Estas teclas son autorrepetitivas, es decir, si las mantiene pulsadas se repite la acción de la tecla. Cuando el cursor llegue al margen derecho, saltará al margen izquierdo de la línea siguiente.

Ahora mantenga pulsadas las teclas SHIFT y CRSR ←/→ y obtendrá el desplazamiento inverso del cursor, de derecha a izquierda. Cuando el cursor llegue al margen izquierdo, saltará al margen derecho de la línea superior.

Con la tecla CRSR ↑/↓ conseguiremos el desplazamiento hacia arriba y hacia abajo del cursor, según si, al mismo tiempo, mantiene pulsada o no la tecla SHIFT. Esto es, con CRSR ↑/↓ llevamos el cursor hacia abajo, mientras que con SHIFT y CRSR ↑/↓ desplazamos el cursor hacia arriba. Observe que si llevamos el cursor al tope superior de la pantalla y continuamos manteniendo pulsadas SHIFT y CRSR↑/↓, lo único que conseguiremos es que el parpadeo del cursor sea muy rápido, pero no afectará en nada al contenido de la pantalla. Sin embargo, si desplazamos el cursor al margen inferior de la pantalla y continuamos manteniendo pulsada la tecla CRSR↑/↓, el cursor no se moverá de esta posición, pero el contenido de la pantalla se irá desplazando hacia arriba línea a línea e irá desapareciendo por el margen superior.

Ahora ya sabemos colocar el cursor en el lugar de la pantalla que deseemos para poder escribir allí lo que queramos. Observe que el paso del cursor por alguna línea que ya esté escrita no afecta en nada al contenido de la línea.

Tecla CLR/HOME

Una vez que hemos ensuciado suficientemente la pantalla, es necesario disponer de algún dispositivo que nos permita limpiarla de una forma cómoda. Quizá en este momento no aprecie mucho su utilidad, pero aprenda su funcionamiento con la seguridad de que será muy útil para realizar trabajos con una buena presentación.

Si pulsa la tecla CLR/HOME, el cursor se coloca en la esquina superior izquierda, pero no se borra nada de lo que tengamos escrito en la pantalla. Pulsando a la vez las teclas SHIFT y CLR/HOME se borra la pantalla y el cursor se sitúa en la esquina superior izquierda.

Tecla INST/DEL

Uno de los hechos a los que tendremos que acostumbrarnos, en nuestra relación con la máquina, es que en muchas ocasiones nos equivocaremos y, por tanto, tendremos que aprender de qué medios disponemos para corregir nuestros errores.

DEL es la abreviatura de DELETE (BORRAR), y al pulsar esta tecla borraremos el carácter que esté situado inmediatamente a la izquierda del cursor. Para eliminar una letra o palabra, dentro de un texto que ya hayamos escrito, tendremos que colocar el cursor, mediante las teclas CRSR, encima del carácter que esté inmediatamente a la derecha del que, o de los que, queremos borrar. Una vez en esta situación, pulse la tecla DEL tantas veces como caracteres desee eliminar. ¡Cuidado! La tecla DEL es autorrepetitiva y, por tanto, si la mantenemos pulsada puede que borremos mucho más de lo necesario.

Suponga, para ir practicando, que tras borrar la pantalla (SHIFT y CLR/HOME) hemos colocado el cursor en medio de ésta, mediante las teclas CRSR, y tecleamos la frase:

MI BUEN AMIGO PEPE

Al final tendremos el cursor tras la última E de PEPE. Pero tras escribir la frase pensamos que sería preferible borrar la palabra BUEN. Para ello mantendríamos pulsada la tecla SHIFT y pulsaríamos la tecla CRSR←/→ hasta colocar el cursor encima de la letra A. Ahora tendremos que pulsar la tecla DEL cinco veces (cuatro por la palabra BUEN y una por el espacio que hay entre BUEN y AMIGO). Observará que la acción de borrar se realiza de derecha a izquierda y que los caracteres situados a la derecha del cursor se irán desplazando hacia la izquierda según vayamos pulsando la tecla DEL, esto es, no queda ningún espacio en blanco por medio. Tras esto, si quiere volver a la posición inicial, pulse la tecla CRSR←/→, para colocarse detrás de la palabra PEPE.

INST representa a la palabra INSERT (INSERTAR) y la utilizaremos para insertar caracteres, que se nos hayan olvidado, entre otros caracteres que ya tengamos escritos. Volviendo con el ejemplo anterior, tras borrar la palabra BUEN, en la pantalla tenemos la frase:

MI AMIGO PEPE

Y ahora pensamos que en realidad sí queríamos poner la palabra BUEN y deseamos volver a colocarla en su sitio. Utilice las teclas CRSR para volver a colocar el cursor sobre la letra A. Mantenga pulsada la tecla SHIFT y pulse la tecla INST tantas veces como caracteres desee insertar, en nuestro caso cinco (cuatro de BUEN y un espacio en blanco que necesitaremos entre BUEN y AMIGO). Note que el cursor no se mueve de su sitio, son los caracteres que hay a su derecha los que se van desplazando, dejando sitio para los caracteres que vamos a introducir. Teclee ahora la palabra BUEN y pulse una sola vez la barra espaciadora (esa tecla larga situada en el borde inferior del teclado) para conseguir el espacio necesario. En este momento el cursor estará sobre la letra A y podemos utilizar la tecla CRSR←/→ CRSR←/→ para situarlo al final de la línea. Si tras introducir la palabra BUEN, en lugar de pulsar la tecla espaciadora, hubiéramos pulsado la tecla, para desplazar el cursor hacia la derecha, entre las palabras BUEN y AMIGO aparecerá un caracter extraño, de la forma [❏], en lugar del espacio en blanco deseado. Esto se debe a que al haber insertado el espacio para cinco caracteres el ordenador estaba esperando aún otro carácter, que muy bien puede ser el espacio en blanco que se obtiene con la barra espaciadora. Al pulsar la tecla CRSR←/→ en lugar del espaciador, el ordenador tomará el carácter [❏], que es como representa la acción del desplazamiento hacia la derecha. Pero, ¡cuidado con el espaciador! Si una vez colocado sobre

la letra A continúa pulsando el espaciador, en lugar de la tecla de desplazamiento, borrará todas las letras por las que pase. No trate todavía de entender bien esto, pues ya lo iremos viendo a medida que avancemos dentro del mundo de nuestro ordenador. De momento, lo más importante es que practique las acciones DEL e INST, hasta llegar a dominarlas. Pruebe a borrar e introducir diversas veces las palabras BUEN y AMIGO, o a crear y corregir otros textos que se le vayan ocurriendo. Cuando desee limpiar la pantalla utilice las teclas SHIFT y CLR.

Impresión de caracteres

Ya hemos utilizado algunos de los caracteres que podemos obtener de nuestro Commodore. Vamos ahora a revisar, de una forma metódica, la amplia gama de caracteres que podemos utilizar.

TECLAS DE LETRAS

«Modalidad mayúsculas»: Recuerde que para pasar de «Modalidad mayúsculas» a «Modalidad minúsculas» ha de pulsar una sola vez y al mismo tiempo las teclas **C=** y SHIFT.

— Pulsando solamente las teclas de letra obtendremos la versión en mayúsculas de la letra correspondiente.
— Pulsando SHIFT y la tecla de letra obtendremos, de los dos caracteres gráficos situados en la falda de la tecla, el de la derecha.
— Pulsando **C=** y la tecla de letra obtendremos el carácter gráfico de la izquierda.

«Modalidad minúsculas»:

— Pulsando directamente una tecla conseguimos la versión minúsculas de la letra.
— Pulsando SHIFT y la tecla aparecerá la versión mayúscula de la letra.
— El resultado de pulsar simultáneamente **C=** y la tecla de letra es el mismo que en *«Modalidad mayúsculas»*.

Observe que si tras escribir diversos caracteres en cierta modalidad pasa a la otra, los caracteres cambiarán a los que correspondan a dicha modalidad.

Teclas con los símbolos +, –, £, @, *,

«Modalidad mayúsculas»:

— Pulsando sólo la tecla obtenemos el símbolo.
— Pulsando al mismo tiempo SHIFT obtenemos el carácter gráfico de la derecha.
— Pulsando al mismo tiempo **C=** aparecerá el carácter gráfico de la izquierda.

«Modalidad minúsculas»: Los resultados son los mismos que en la anterior modalidad, exceptuando que en *«Modalidad minúsculas»* los caracteres y de las teclas £ y * se convierten en rayados diagonales del cuadradito, y el carácter de la tecla @ se convierte en el carácter

Tecla ↑/π

«Modalidad mayúsculas»:

— Pulsando directamente la tecla obtenemos el símbolo.
— Si al mismo tiempo pulsamos SHIFT o **C=** aparece la letra π.

«Modalidad minúsculas»: La única diferencia es que el símbolo π pasa a ser un reticulado de la posición de carácter.

Teclas = y ←

— Siempre producen el símbolo impreso encima de la tecla.

Teclas [,], <, > y ?

«Modalidad mayúsculas»:

— Pulsando sólo la tecla obtenemos el símbolo inferior.
— Si además pulsamos SHIFT o **C=** aparece el símbolo superior.

«Modalidad minúsculas»: Funciona igual que la *«Modalidad mayúsculas»*.

TECLAS NUMERICAS

De estas teclas vamos a ver sólo dos posibilidades. La acción conjunta de **C=** ó CTRL con estas teclas la estudiaremos en el apartado siguiente: «Obtención de colores».

«Modalidad mayúsculas»:

— Pulsando directamente la tecla obtenemos el número correspondiente.
— Si al mismo tiempo pulsamos SHIFT obtenemos el símbolo que aparece impreso en la parte superior de la tecla.
Una excepción es la tecla del cero, Ø, que al no tener ningún símbolo impreso origina el carácter Ø, tanto si utilizamos SHIFT como si no. (El número cero lo representaremos siempre por Ø, para distinguirlo de la letra O.)

«Modalidad minúsculas»: Funciona igual que la *«Modalidad mayúsculas»*.

> Si, tras pulsar las comillas « que aparecen en la tecla del 2, trata de borrar la pantalla SHIFT y CLR, verá que aparece un símbolo en forma de corazón, pero que la pantalla no se borra. Asimismo, si trata ahora, tras pulsar un símbolo de comillas, de desplazar el cursor, no conseguirá este efecto, sino que aparecerán ciertos símbolos que para el ordenador representan las acciones de desplazamientos del cursor. Y lo mismo pasa con todas las teclas de acción, menos para la tecla DEL (borrar). Además, aunque utilice DEL y borre las comillas, seguirá pasando lo mismo, el ordenador recuerda que pulsó unas comillas.

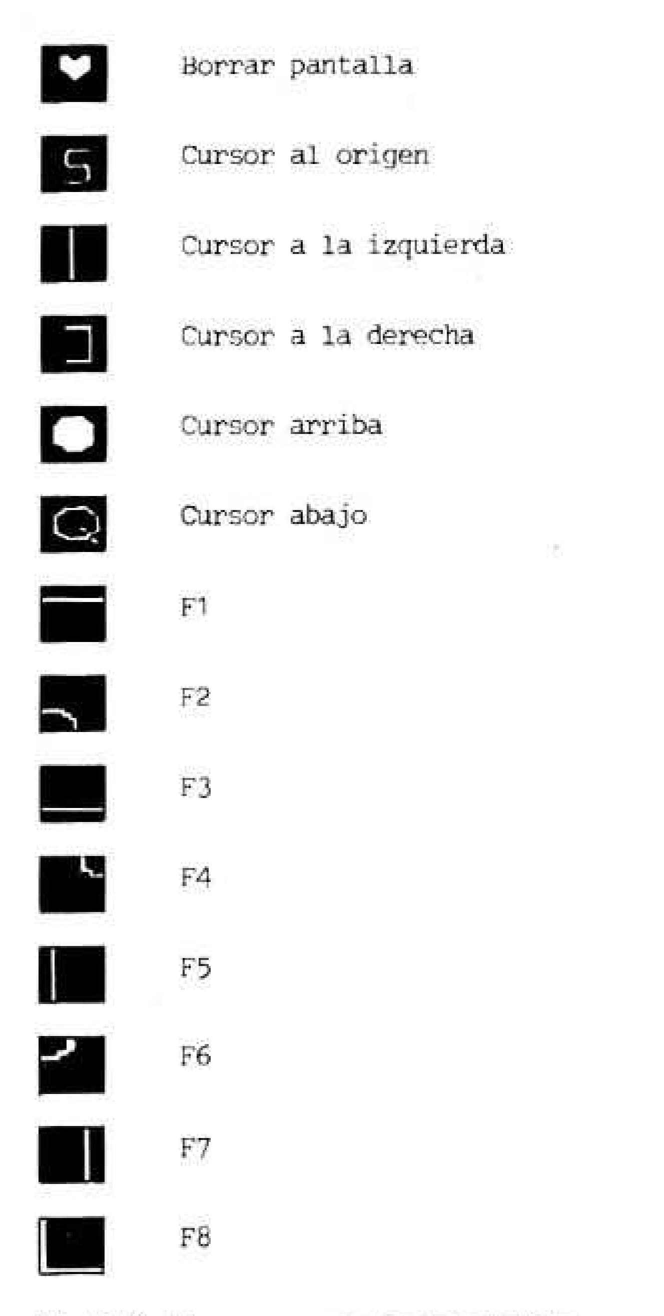


Fig. 1.7. Símbolos que representan funciones especiales.

> En la figura 1.7 se exponen los símbolos que aparecen al pulsar, tras unas comillas, ciertas teclas de funciones especiales. La forma más práctica de salir, en este momento, de esta situación es apagando y volviendo a encender el ordenador. No se preocupe por tratar de entender este comportamiento. No significa que su máquina se haya estropeado. El Commodore 64 funciona así, y poco a poco nos iremos familiarizando con su comportamiento. Otras formas de salir de la situación anterior son: tecleando otro símbolo de comillas (el número de símbolos ha de ser par), el ordenador vuelve a su situación normal; pulsando la tecla RETURN; pulsando a la vez las teclas RESTORE y RUN/STOP.

Escribiendo en colores

La gama de colores que podemos obtener de nuestro Commodore es muy amplia, 16 colores diferentes. Si ya nos hemos cansado de que el cursor sea azul claro, podemos mantener pulsada la tecla CTRL o la tecla **C=** y pulsar una de las teclas numéricas del 1 al 8. Hágalo diversas veces y observe la variedad de colores que nos ofrece el ordenador. Ahora podremos ir escribiendo en tinta amarilla, verde, roja, etc. (ver la Fig. 1.8). Así, si pulsamos CTRL y 8, el cursor pasa a amarillo, y cada vez que pulsemos una tecla de este carácter aparecerá en tinta amarilla. Para devolver al cursor su color original pulse **C=** y 7.

Tecla \ *con*	*CTRL*	**C=**	*Abreviatura inglesa*
1	Negro	Naranja	BLK
2	Blanco	Marrón	WHT
3	Rojo oscuro	Rojo claro	RED
4	Azul verdoso	Gris oscuro	CYN
5	Púrpura	Gris intermedio	PUR
6	Verde oscuro	Verde claro	GRN
7	Azul*	Azul claro**	BLU
8	Amarillo	Gris claro	YEL

Fig. 1.8. Posibles colores del cursor.

* Color normal del fondo.
** Color normal del cursor.

Inversión de los colores del papel y de la tinta

Cuando escribimos normalmente, los caracteres aparecen en el color que le hayamos asignado a la tinta, sobre un fondo de color azul oscuro. Esto podemos invertirlo utilizando la acción RVS ON que aparece en la tecla del 9. Pulse CTRL y 9, y a partir de este momento todos los caracteres aparecerán con tinta azul oscura sobre un fondo del color que antes tuviéramos asignado a la tinta.

Para salir de esta situación pulse CTRL y Ø (RVS OFF) y los caracteres irán apareciendo en su forma normal (Fig. 1.9).

> Si estamos dentro de un símbolo de comillas, estas asignaciones de colores e inversiones de los colores de tinta y fondo no aparecerán en su forma normal, sino como caracteres que el C-64 entiende (ver Fig. 1.7). Pero, como ya hemos indicado, no se preocupe todavía de esto.

	Tecla	*Abreviatura*	*Acción*
CTRL	9	RVS ON	Colores de fondo y tinta invertidos
CTRL	∅	RVS OFF	Vuelta a la apariencia normal

Fig. 1.9. Inversión del aspecto de los caracteres.

La tecla CTRL

CTRL es la abreviatura de la palabra CONTROL. Se sorprenderá de la diversidad de acciones de control que puede realizar con esta tecla. En el apartado anterior vimos cómo utilizar CTRL para controlar el color y la apariencia de los caracteres, y en la figura 1.10 se exponen otras acciones de control.

	Tecla	*Acción*
CTRL	H	Inhibe el cambio de «Modalidad mayúsculas» a «Modalidad minúsculas», y viceversa.
CTRL	I	Neutraliza el efecto de CTRL H.
CTRL	M	Igual que la tecla RETURN. No se preocupe, enseguida veremos la tecla RETURN.
CTRL	N	Pasa de «Modalidad mayúsculas» a «Modalidad minúsculas».
CTRL	Q	Igual que CRSR ↓.
CTRL	:	Igual que CRSR →.
CTRL	R	Igual que RVS ON.
CTRL	S	Igual que HOME.
CTRL	T	Igual que DEL.
CTRL	E	Color de tinta blanco.
CTRL	£	Color de tinta rojo.
CTRL	↑	Color de tinta verde.
CTRL	=	Color de tinta azul oscuro.

Fig. 1.10. Algunas acciones de CONTROL.

La tecla RETURN

Siempre que haya escrito algo y quiera que el ordenador lo lea, ha de pulsar la tecla RETURN. Mientras que no pulse esta tecla, el ordenador no hará caso de lo que esté escribiendo en la pantalla. Cuando la pulse, el ordenador leerá y tratará de interpretar lo que usted haya escrito. Como todavía no hemos aprendido el lenguaje del C-64, lo más probable es que obtenga el mensaje de SYNTAX ERROR: error de sintaxis.

La tecla RUN/STOP

Si estamos ejecutando un programa en el ordenador y pulsamos esta tecla, inmediatamente se detiene la ejecución del programa. Esta acción la entenderemos mejor en el capítulo próximo.

La acción combinada de las teclas SHIFT y RUN/STOP provoca que el C-64 cargue y ejecute automáticamente un programa almacenado en la Datassette.

> No hay que confundir esta tecla RUN/STOP con el comando RUN del BASIC que veremos más adelante.

La tecla RESTORE

Ha de utilizarse conjuntamente con la tecla RUN/STOP y su empleo interrumpe cualquier programa bajo ejecución y restablece todas las condiciones internas de la máquina a su estado original, tal y como si acabáramos de encender el ordenador. Así, el color de la pantalla vuelve a ser de azul claro sobre fondo azul oscuro, se interrumpe cualquier sonido que estuviera emitiendo el C-64, etc.

Pero la gran ventaja de utilizar RESTORE y RUN/STOP, en lugar de apagar y volver a encender el ordenador, es que no perdemos el programa que estuviera en memoria y podemos corregirlo, mejorarlo o volver a ejecutarlo.

> No hay que confundir esta tecla RESTORE con el comando RESTORE del BASIC, que veremos más adelante.

Teclas de función programables

Reciben este nombre las cuatro teclas situadas en el margen derecho del C-64 y etiquetadas con los caracteres f1/f2, f3/f4, f5/f6, y f7/f8. A diferencia de otros ordenadores, en el Commodore estas teclas vienen sin ninguna función asignada, y los métodos para que el usuario pueda asignarles alguna función (distinta del mero reconocimiento de si se ha pulsado una de estas teclas o no) se basan en el empleo del código máquina, y su explicación queda fuera del alcance de este libro. Sin embargo, algunos de los programas preparados para el C-64 sacan partido de estas teclas de función.

HABLANDO CON MI ORDENADOR

En este momento ya hemos aprendido a utilizar el teclado del C-64 y sólo nos queda un punto para poder empezar a sacarle partido: *aprender el lenguaje que nuestro ordenador entiende*.

El lenguaje que viene instrumentado en el Commodore es el BASIC y, co-

mo cualquier otro lenguaje, se compone de un conjunto de palabras, llamadas comandos, palabras clave etc., y de un conjunto de reglas sintácticas. El único problema, a diferencia de cuando tratamos de entendernos con nuestro profesor de inglés, es que el ordenador no puede tratar de entendernos. Para él los inteligentes somos nosotros y no hace más que obedecernos. Así que o escribimos correctamente las palabras y utilizamos adecuadamente la sintaxis o no podrá mos entendernos. ¡Pero no se asuste! El conjunto de palabras y normas sintácticas no es muy grande y podemos ensayarlas poco a poco, según vayamos adquiriendo confianza. Cuando termine este libro podrá utilizar combinaciones sofisticadas para sacarle el máximo provecho a sus programas.

Los comandos que hemos de aprender a utilizar son del tipo:

ORDEN «contenido de la orden»

Es decir, le damos una orden al computador y tras la orden le escribimos con qué tiene que llevarla a cabo. Así, ejemplos típicos de órdenes son:

PRINT que significa «imprime»,
LIST que significa «saca un listado de lo que tienes dentro»,
RUN que significa «ejecuta un programa»,

etcétera. Quizá el método más fácil para aprender a darle órdenes al computador sea utilizándolo como si fuera una calculadora. Vamos a probar.

El ordenador como calculadora

Acabamos de mencionar la palabra PRINT, que indica al ordenador que debe escribir algo en la pantalla. Así, por ejemplo, para que escriba un tres tecleamos

PRINT 3

y pulsamos la tecla RETURN. La tecla RETURN es la que utilizamos para indicarle al computador que ya hemos terminado de escribir y que queremos que lea lo que hemos escrito. Mientras no pulse RETURN no pasará nada. Nada más pulsar esta tecla el ordenador leerá la línea, comprobará que es correcta y si así ocurre obedecerá la orden dada. En nuestro caso aparecerá un 3 en la pantalla, debajo de la línea PRINT 3. Más abajo aparecerá el mensaje READY con el que la maquina nos comunica que está preparada para la orden siguiente. Probemos con la línea:

PRINT 3+4

(acuérdese de pulsar RETURN). En este caso en la pantalla no aparece 3+4, sino un 7. Al llegar al contenido del PRINT y encontrarse con una operación, el ordenador interpreta que deseamos que la realice y nos imprime el resultado. Los símbolos aritméticos que podemos utilizar con el C-64 son: + para la adición, — para la sustracción, * para la multiplicación, / para la división, y ↑ para la exponenciación. Así, las órdenes siguientes producirán los resultados:

```
PRINT 2+3*4—1        imprime 13
PRINT 2/4+6*3/2      imprime 9.5
PRINT 2↑3/2+5*3      imprime 19
```

Observe que para separar los enteros de los decimales se utiliza un punto, y no una coma como es habitual en España. Lo siento, pero tendrá que acostumbrarse a esto, pues si no el ordenador no le entenderá y se lo indicará con un mensaje de error de sintaxis (Syntax error). Además, no puede poner puntos ni comas para indicar miles o millones.

No siempre es fácil analizar cómo hemos de escribir una expresión aritmética para obtener el resultado deseado. Por ejemplo, en la última de las expresiones anteriores nos encontramos con la operación 2↑3/2. ¿Cómo se lee esto? Podría ser bien $2^{3/2}$, ó $2^3/2$. Para resolver este problema recuerde que el ordenador siempre actúa bajo dos reglas.

1: Resolver en primer lugar las operaciones de mayor prioridad (ver Fig. 1.11).
2: Dentro de un mismo nivel de prioridad, las operaciones se efectúan de izquierda a derecha.

Así la expresión 2↑3/2 se interpreta como $2^3/2$.

Operador	*Significado*	*Prioridad*
()	Paréntesis	9
↑	Exponenciación	8
—	Signo menos (números negativos)	7
*	multiplicación	6
/	División	6
+	Adición	5
—	Sustracción	5

Fig. 1.11. Operadores aritméticos y sus prioridades.

Las prioridades de nivel 4, 3, 2 y 1 corresponden a los operadores relacionales y a los operadores de Bool, que veremos más adelante.

Aun con esto no es fácil expresar ciertas operaciones aritméticas ¿Cómo escribiríamos $\frac{3+2}{5}$ ó $\frac{3}{2+1}$? Es obvio que las expresiones 3+2/5 y 3/2+1 son incorrectas, pues sus resultados respectivos son 3.4 y 2.5, mientras que los de las expresiones originales son ambos 1. Para resolver este problema se recurre a la utilización de los paréntesis. Y aquí tenemos otras dos reglas a tener en cuenta en las operaciones.

1: Siempre se evalúa primero lo que esté encerrado entre paréntesis.
2: Si unos paréntesis contienen otros paréntesis, se evalúan en primer lugar los paréntesis más internos.

De este modo la operación $\frac{3+2}{5}$ se expresa como (3+2)/5, y la operación $\frac{3}{2+1}$ se puede expresar como 3/(2+1).

Notación científica

Hasta aquí hemos representado los números mediante lo que se llama «notación normal». Un número en «notación normal» puede ser un número entero o un número fraccionario (con un punto decimal), precedido de un signo +, si es positivo, o de un signo −, si es negativo. Ante la ausencia de signo se interpreta que es positivo. Así, ejemplos de números en notación normal son:

Enteros	*Fraccionarios*
27	16.377
−14	−Ø.1422
42ØØ	−3.7
Ø	Ø.ØØ15

Los números en notación normal pueden tener al menos ocho cifras decimales de precisión, e incluso pueden llegar a tener 9, dependiendo del número. El BASIC del C-64 realiza un redondeo automático y, generalmente, redondea por encima si el primer dígito despreciado es mayor que cinco y redondea por debajo si éste es menor o igual que cuatro. Sin embargo, hay excepciones, por ejemplo:

```
PRINT .4444444446   imprime Ø.444444445
PRINT .4444444445   imprime Ø.444444445
PRINT .4444444444   imprime Ø.44444444
```

Hay excepciones. Trate de encontrar alguna. En notación normal el C-64 puede representar números comprendidos entre el ØØ1 y el 999999999 (bien sean positivos o negativos), pero los números fuera de este rango los representa mediante la llamada «notación científica». En notación científica un número se representa mediante una expresión del tipo

«Mantisa» E «Signo» «Exponente»

Mantisa: Es un número en notación normal.
E: Se utiliza esta letra para separar la mantisa del exponente.
Signo: Indica si el exponente es negativo o positivo. Ante la ausencia de signo se asume que el exponente es positivo.
Exponente: Número entero formado por una o dos cifras. Indica el número de lugares que hay que trasladar hacia la derecha (si el exponente es positivo) o hacia la izquierda (si es negativo) el punto decimal de la mantisa para obtener la notación normal.

Por ejemplo:

Notación científica	*Notación normal*
6.3 EØ5	63ØØØØ
1.2 E-Ø7	Ø.ØØØØØØ12
43 E-Ø2	Ø.43
-43 E-Ø2	-Ø.43
43 E+Ø2	43ØØ

No se asuste si no entiende bien la notación científica. Se han escrito miles y miles de programas muy útiles sin tener que preocuparse de esta notación.

Aun utilizando la notación científica existe un límite a los números que puede manejar el Commodore.

El número mayor es 1.70141183 E+38.
El número menor es 2.93873588 E−39.

Bien sean positivos o negativos, cualquier número que sobrepase el límite superior originará un mensaje de error «OVERFLOW ERROR» (número demasiado grande).

Cualquier número que sobrepase el límite inferior dará como resultado un cero.

PROGRAMANDO EL ORDENADOR

La capacidad más interesante del ordenador es que podemos almacenar órdenes en su memoria y después ejecutarlas cuando lo deseemos. Hasta ahora todo lo que hemos hecho ha sido darle órdenes directas, tales como PRINT 25, y el ordenador las ejecuta y las olvida. Es decir, si nos hemos equivocado en un cálculo, o necesitamos volver a hacer una operación, tendremos que teclear la línea entera. Trabajar así es lo que recibe el nombre de «modo directo».

Para entrar en «modo programado» tenemos que escribir un número delante de la orden que deseamos darle al computador.

Así, teclee:

```
1Ø PRINT 25
```

y cuando pulse RETURN observará que el ordenador no ejecuta esta orden, sino que la guarda en su memoria, y el cursor pasa a la línea siguiente.

Teclee ahora:

```
2Ø PRINT 43-27
```

y al pulsar RETURN vuelve a suceder lo mismo. Pulse las teclas SHIFT, y CLR/HOME para borrar la pantalla. Teclee la orden directa RUN (no, no pulse la tecla RUN/STOP, debe teclear las letras de la palabra clave RUN). Esta es la forma de mandarle al ordenador que ejecute el programa que tiene dentro, pulse RETURN y verá aparecer los resultados de la ejecución: los números 25 y 16.

Las líneas 1Ø y 2Ø siguen en la memoria de la máquina y podemos ejecutarlas tantas veces como queramos. Cuando deseemos volver a ver nuestro progra-

ma, basta con teclear la orden directa LIST y pulsar RETURN. LIST significa: «Saca un listado del programa que tienes en la memoria».

Añadamos ahora las líneas siguientes:

```
15 PRINT "43-27"
30 PRINT "HE AQUI LA DIFERENCIA"
40 PRINT "FACIL, ¿NO?"
 5 PRINT "MI PRIMER PROGRAMA"
```

(Acuérdese de pulsar RETURN tras cada línea.) Pulse SHIFT y CLR/HOME y liste el programa, LIST y RETURN. Verá que, aunque las líneas 15 y 5 se han introducido posteriormente, en el listado del programa aparecen todas las líneas ordenadas crecientemente por su número. Este número delante de las órdenes recibe el nombre de «número de línea», y en el C-64 puede utilizar números comprendidos entre el 0 y el 63999. Sin embargo, es práctica habitual utilizar números de diez en diez; así 10, 20, 30, ... Si lo desea, puede numerar las líneas utilizando la secuencia 1, 2, 3,... etc., pero si se dejan libres algunos números de línea, esto le da la oportunidad de intercalar líneas en los programas, lo cual es muy útil cuando nos damos cuenta de que para que funcione es necesario introducir algunas líneas entre otras ya existentes. Si usted desea numerarlas consecutivamente, o de cinco en cinco, o de diez en diez, o etc., éste es su gusto; el único caso desaconsejable es el primero.

En la figura 1.12 se recogen las diversas formas de utilizar el comando LIST.

Formato	*Acción*
LIST	Lista el programa completo.
LIST 30	Lista la línea 30.
LIST —30	Lista desde la primera línea hasta la 30.
LIST 30—	Lista desde la línea 30 hasta el final.
LIST 15-40	Lista desde la línea 15 hasta la 40.

Fig. 1.12. Diversas posibilidades de utilización de LIST.

Así, el formato general de la sentencia LIST es:

LIST «número 1» — «número 2»

Número 1: Todas las líneas cuyo número sea superior a «número 1» aparecerán en el listado a menos que sobrepasen el valor indicado en «número 2». Por defecto el listado comenzará en la primera línea el programa.

Número 2: Todas las líneas cuyo número sea inferior a «número 2» aparecerán en el listado a menos que sea también inferior a «número 1». Por defecto el listado terminará en la última línea del programa.

Si tras LIST aparece un solo número (sin guión), se listará la línea correspondiente a ese número, si es que existe.

Para ejecutar el programa anterior teclee RUN y pulse RETURN. En la pantalla aparecerá:

```
MI PRIMER PROGRAMA
25
43-27
16
HE AQUI LA DIFERENCIA
FACIL, ¿NO?
```

y ya sabemos cómo imprimir literales. Observe el resultado de la línea 5. Cuando el ordenador llega al contenido del PRINT y detecta las comillas, interpreta que deseamos que el contenido entre comillas aparezca en pantalla tal y como está, y así lo hace. Es decir, cuando utilizamos comillas le indicamos al ordenador que copie exactamente lo que hay entre comillas. Fíjese ahora en la diferencia entre las líneas 15 y 20

```
15 PRINT "43-27"    imprime 43-27
20 PRINT 43-27      imprime 16
```

Esta diferencia la reflejamos diciendo que en la línea 20 estamos trabajando con contenidos numéricos, mientras que en la 15 el contenido es alfanumérico. Por alfanumérico nos referimos a cualquier secuencia de letras, dígitos o símbolos, tales como el punto, la coma, etc. (esto suele recibir el nombre de «cadena»). Y para que el ordenador trate con ellos deben ir encerrados entre comillas.

Podemos no poner las comillas finales, por ejemplo:

```
100 PRINT "Hola
```

y el ordenador interpretará bien la línea, pero no se acostumbre a hacer esto, pues sólo lo podrá utilizar en PRINT cuando sean tan sencillos como el del ejemplo anterior. En sentencias PRINT más complicadas, por ejemplo, con varios campos, obtendrá mensajes de error.

Otra forma de utilizar la instrucción RUN es escribiendo detrás un número de línea. Así RUN solo, sin ningún número detrás, ejecuta el programa completo, empezando por la línea 5 y terminando en la 40, mientras que el comando RUN 30 ejecuta desde la línea 30 hasta el final. La estructura general de la sentencia RUN es:

RUN «núm. de línea»

Número de línea: Indica la línea de programa en que comenzará la ejecución. Por defecto se toma desde la primera línea. Si el número indicado no coincide con alguna línea del programa, obtendremos el mensaje de error «UNDEF'D STATEMENT ERROR».

Corrección de programas

Con frecuencia, escribiendo programas, cometeremos muchos errores, y tendremos que conocer de qué métodos disponemos para corregirlos. Vamos a repasar aquí las posibilidades que nos ofrece el Commodore.

Suponga, en primer lugar, que en la línea 5 deseábamos poner:

5 PRINT "MI SEGUNDO PROGRAMA"

en lugar de:

5 PRINT "MI PRIMER PROGRAMA"

Este es el momento de volver a repasar el funcionamiento de las teclas CRSR ←/→, CRSR ↑/↓ e INST/DEL. Liste el programa y utilice las teclas CRSR para colocar el cursor SOBRE la P de PROGRAMA. Utilice la tecla DEL para borrar la palabra PRIMER. Ahora pulse mantenidamente SHIFT y pulse ocho veces la tecla INST/DEL para insertar los ocho espacios que necesitamos (siete para SEGUNDO y un espacio en blanco entre las palabras). Teclee la palabra SEGUNDO y pulse una vez la barra espaciadora para colocarse encima de la P. Pulse ahora la tecla RETURN para que el ordenador lea la línea corregida.

> **Si después de corregir, no pulsa RETURN, sino que utiliza las teclas CRSR para salir de esta posición, el ordenador no tomará en cuenta la corrección, aunque ésta aparezca en pantalla. Compruébelo actuando de este modo e introduciendo después el comando LIST.**

Si lo que deseamos es borrar una de las líneas existentes, basta con teclea el número de línea correspondiente y pulsar la tecla RETURN.

Por ejemplo: pulse 1Ø y RETURN. Ahora, liste el programa y comprueb que la línea 1Ø ha desaparecido.

Para cambiar la línea

2Ø PRINT 43-27

para que el contenido sea 5*4, en lugar de 43-27, podemos utilizar la tecl INST/DEL para realizar las correcciones o simplemente teclear

2Ø PRINT 5*4

y pulsar RETURN. La nueva línea 2Ø ocupará el lugar de la antigua. Com pruébelo listando el programa. A veces es más cómodo escribir de nuevo un línea que andar buscándola y corrigiéndola. En ocasiones es necesario escrib varias líneas cuyo contenido es bastante similar. En este caso es más cómod componer las diversas líneas a partir de la primera de ellas. Suponga que, e este caso, deseamos duplicar la línea 4Ø. Utilice las teclas CRSR e INST/DE para borrar el 4 y poner un 5 en su lugar. Pulse RETURN y después liste programa. Tendremos las líneas:

4Ø PRINT "FACIL, ¿NO?"
5Ø PRINT "FACIL, ¿NO?"

y sólo con cambiar un carácter. Fácil, ¿no?

Cuando se canse de jugar con este programa, o con cualquier otro de los iniciales, introduzca (RETURN) el comando NEW. NEW significa «nuevo» y borra el programa de la memoria de la máquina. Si ahora trata de listar el programa, sus esfuerzos serán inútiles.

MAS DETALLES SOBRE PRINT

Con una sentencia PRINT podemos imprimir más de un número o una cadena. Para ello hemos de utilizar «separadores», entre los números o cadenas.

Separadores

Vamos a ver aquí la acción del punto y coma (;) y de la coma (,). Considere el programa siguiente:

```
1Ø PRINT 34;5
2Ø PRINT "2+3=";2+3
3Ø PRINT "2+3=";
4Ø PRINT 2+3
5Ø PRINT "EL RESULTADO DE";2;"+";3;"ES"
   ;2+3
6Ø PRINT "HOLA";"PEPE"
```

En la línea 1Ø el ordenador se encuentra con dos campos numéricos separados con un punto y coma. Observe que los imprime uno a continuación del otro, pero separados por dos espacios en blanco. El primero lo pone el Commodore como separación de campo numérico; el segundo corresponde al signo, en este caso no impreso del número 5. En el resultado de la línea 6Ø comprobará que en caso de que los campos sean alfanuméricos, el C-64 no deja espacios en blanco. En la línea 2Ø se separan, mediante un punto y coma, un campo alfanumérico de uno numérico. Pero fíjese en las líneas 3Ø y 4Ø. Tratan de demostrar que el punto y coma no sólo sirve de separador de campos, sino que además obliga a que lo siguiente que se imprima en pantalla aparezca a continuación, sin salto de línea. Así, un PRINT que termina en punto y coma inhibe la acción natural de salto de línea del PRINT siguiente. En la línea 5Ø se presenta un PRINT con seis campos de impresión. Un exceso. A este respecto, la única limitación es que una línea de programa, en el C-64, no puede ocupar más de 80 caracteres (dos líneas de pantalla). Si no respeta esta norma, el C-64 no hará caso de la línea que pretenda introducir.

El separador coma (,) actúa de forma bastante diferente. Sirve también para separar campos, como el punto y coma, pero además divide la pantalla en cuatro zonas de impresión, cada una de 10 espacios. Para comprender mejor esto mire la figura 1.13. Cada fila de la imagen está dividida en 40 posiciones de carácter (numeradas del Ø al 39). Si introducimos la instrucción:

1Ø PRINT 1, 2, 3, 4

Veremos aparecer los números 1, 2, 3 y 4 en las posiciones 1, 11, 21 y 31 de una fila (recuerde que cuando se imprimen números, siempre hay un espa-

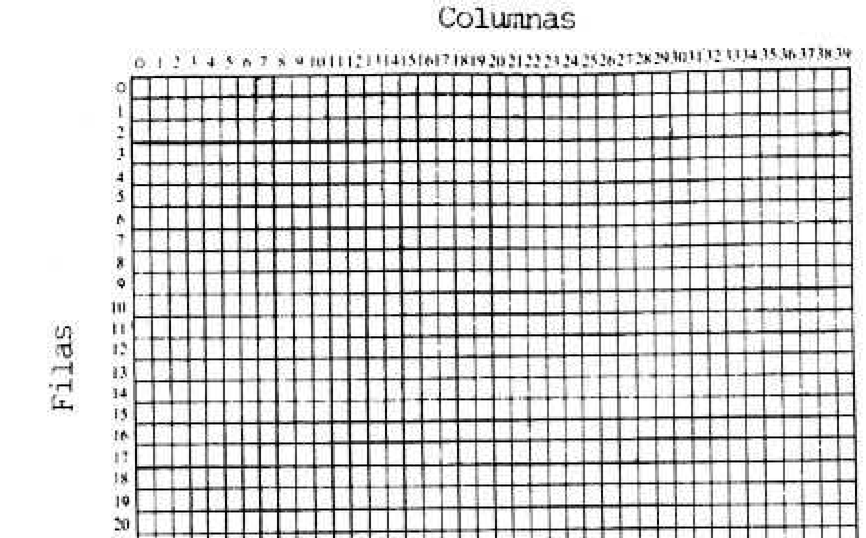


Fig. 1.13. La pantalla del C-64.

cio delante para poner el signo), indicándonos que es ahí donde comienzan la primera, segunda, tercera y cuarta zona de impresión, respectivamente. Así, el efecto de la coma es el de saltar al principio del siguiente cuarto de pantalla. Teclee, por ejemplo, el programa:

```
10 PRINT 1,2,3,4,5,6,7,8
20 PRINT "PEPE","JUAN","LUIS","MANUEL"
30 PRINT "NO VALE TANTO","PEPE"
```

Con las dos primeras líneas observaremos el comportamiento normal de la coma, pero en la línea 30 el primer campo ocupa más de 10 caracteres, por lo que, al saltar al principio de la zona siguiente, PEPE aparecerá impreso a partir de la columna 20. Pruebe a ver qué ocurre si tras el 8 de la línea 10 coloca una, dos, o tres comas. ¿Qué significa el resultado de ejecutar el programa?

Funciones de impresión

Puede que le parezca excesivo tanto truco de impresión, sin embargo, es algo que agradecerá cuando sus programas se vayan complicando y desee una buena presentación visual de la ejecución. Dos de los elementos más típicos en una buena impresión tienen que ver con el margen y con el espacio entre los elementos a imprimir.

Para resolver el problema del margen, el C-64 dispone una función llamada TAB (tabulación). Que ha de ir seguida de un número encerrado entre paréntesis. Este número indica en qué columna se va a imprimir el primer carácter de la expresión deseada (los números de columna, en la pantalla del Commodore van del 0 al 39). Por ejemplo, ensaye el programa siguiente (para borrar el programa anterior, teclee NEW y pulse RETURN):

```
10 PRINT TAB(11);"ESTO ESTA CENTRADO"
20 PRINT
30 PRINT
40 PRINT "ESTO NO"
50 PRINT TAB(5);"Y ESTO";TAB(30);"TAMPOCO"
```

En la primera línea utilizamos TAB(11) para centrar un literal. Recuerde que la pantalla tiene 40 columnas. Por otra parte, la frase ESTO ESTA CENTRADO ocupa 18 caracteres (acuérdese de contar los espacios en blanco). Así, 40-18=22, nos sobran 22 caracteres en la línea, y para centrar la frase tendremos que dejar 11 en el margen derecho y otros 11 en el izquierdo. En las líneas 20 y 30 se utilizan dos PRINT vacíos para provocar que la frase ESTO NO se escriba dos líneas más abajo de la frase anterior. El ordenador al leer un PRINT lo primero que hace es saltar a la línea siguiente. Si no hay nada para imprimir pasa a leer la instrucción siguiente. La línea 50 muestra cómo podemos utilizar diversos TAB en un PRINT. Tenga cuidado de no introducir un espacio en blanco entre el TAB y el primero de los paréntesis, pues obtendrá un mensaje de error. "?BAD SUBSCRIPT ERROR IN «núm. de línea» *x*", que indica que hay un error en el índice del TAB en la línea *x*, núm. de línea *x*.

> El separador «;» puede omitirse tras el TAB y también, en muchos casos, entre campos cuya separación esté bien definida. Por ejemplo, teclee:
>
> 100 PRINT "3+2=" 3+2
>
> y verá que no incurre en error por omitir el «;». Modifique la línea 50 del programa anterior, borrando todos los puntos y coma, y vuelva a ejecutarlo.

Borre el programa anterior NEW y RETURN, limpie la pantalla SHIFT y CLR/HOME y pasemos al siguiente ejemplo:

```
10 PRINT TAB(5)"DATO";TAB(20)"OTRO";TAB(
   30)"DATO"
20 PRINTSPC(5)"DATO";SPC(11)"OTRO";SPC(6
   )"DATO"
```

Ejecute el programa y observe que el efecto de ambas líneas es el mismo. En las líneas 10 utilizamos TAB para indicar la columna en que deseamos que se comience a escribir cada cadena, mientras que en la línea 20 empleamos SPC (abreviatura de SPACE, ESPACIO) para indicar el número de espacios en blanco que deseamos que existan antes de escribir cada cadena.

Dentro de los paréntesis del TAB o del SPC podemos poner un número entre 0 y 255. Un número fuera de este rango provocará un mensaje de "?ILLEGAL QUANTITY ERROR IN «núm. de línea»", que significa: error por cantidad no permitida en la línea indicada por «núm de línea».

Utilización de las teclas CLR/HOME y CRSR dentro del PRINT

Las acciones de borrar la pantalla CLR y de desplazamiento del cursor HOME y CRSR puede emplearse dentro de una sentencia PRINT. Esto es muy útil, pues nos permite borrar la pantalla antes, o en medio, de la ejecución de un programa y colocar los caracteres a imprimir en el lugar que deseemos de la pantalla. Para realizar esto tendremos que pulsar la tecla deseada, por ejemplo, CLR, dentro de un PRINT alfanumérico (aquel cuyo contenido está encerrado entre comillas), pero nos extrañará que al hacer esto no aparecerá la palabra CLR, HOME o CRSR, sino un carácter que representa esta acción.

En la figura 1.14 se exponen los caracteres que representan a estas acciones. (Estos caracteres, así como los que representan a los colores y otras acciones, se expusieron también en la figura 1.7.)

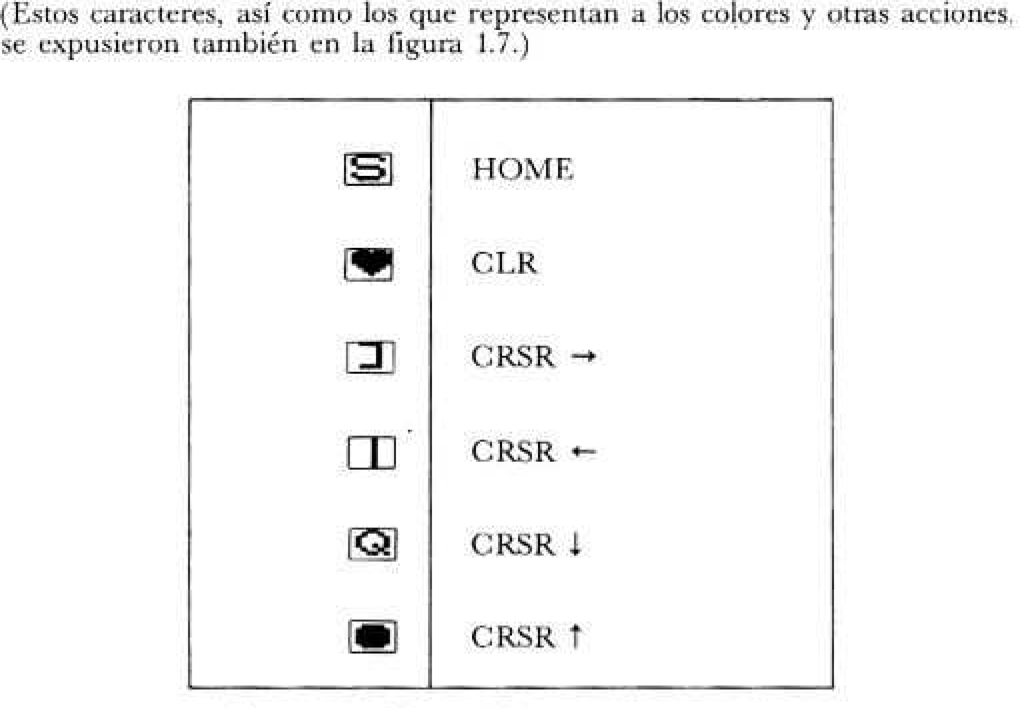


Fig. 1.14. Símbolos especiales que representan acciones.

Así, por ejemplo, teclee el programa siguiente:

```
10 PRINT "♥"
20 PRINT "▶▶▶▶▶QQQQQQQHOLA"
30 PRINT "S"
40 PRINT "▶▶▶▶▶▶▶▶▶▶PEPE"
```

La línea 10 borra la pantalla y coloca el cursor en la esquina superior izquierda. La línea 20 desplaza el cursor cinco columnas a la izquierda, [ ], y siete filas hacia abajo, [Q] y en esta posición escribe la palabra HOLA. Tras esto, la línea 30 devuelve el cursor a la esquina superior izquierda, pero sin borrar la pantalla, y la línea 40, después de desplazar el cursor 10 lugares a la izquierda, imprime la palabra PEPE. Observe que hemos conseguido imprimir el contenido de la línea 40 por encima de la palabra PEPE. Ahora ya sabemos cómo escribir en cualquier lugar de la pantalla. ¡Enhorabuena!

Entre las comillas también podemos introducir los colores y otras acciones, tales como RVS ON, RVS OFF etc. y, como en el caso anterior, en la línea PRINT aparecerán los caracteres que los representan. Volveremos sobre esto al hablar de gráficos.

Para finalizar, vamos a exponer el formato general de la sentencia PRINT:

PRINT «lista de impresión»

Lista de impresión: puede estar formada por diversos campos, alfanuméricos o numéricos, delimitados por separadores.

LA DATASSETTE

Esto es mucho más importante de lo que a primera vista pueda parecer. Poco a poco irá mejorando en su programación en BASIC e irá construyendo programas que es una pena perder cuando desconecte el C-64. *Cuando desconecta el C-64 se borran todas las instrucciones que usted haya almacenado en su memoria.* El trabajo de teclear un programa es demasiado tedioso como para tener que volver a repetirlo cada vez que encendemos el ordenador. Además, otras veces, desearemos copiar y guardar programas útiles que hayamos visto en algún libro, revista, etc. Pues bien, para esto sirve la Datassette. Con ella podremos grabar, como si fuera música, todos los programas que queramos en cintas, y después, cuando al día siguiente queramos volver a ver un programa, podremos reproducirlos en el ordenador a partir de la cinta. ¡Estupendo! ¿No? Poco a poco iremos creando nuestra biblioteca de juegos, utilidades, programas educativos, etc. Pero un poco de calma, lo primero que hay que hacer es aprender a utilizarla. Para empezar, unos consejos.

— Compre cintas de corta duración, C-5, C-10, C-15, C-20. En ocasiones arriesgadas se pueden utilizar cintas de 60 y 90 minutos, pero cuanto más largas, más posibilidades de errores, y posible deterioro, y más tiempo gastado en encontrar los programas. Claro que también son, proporcionalmente, más baratas, pero el riesgo no merece la pena.
— Rebobine varias veces las cintas, en ambas direcciones, antes de utilizarlas. Las cintas se bobinan a alta velocidad, con lo cual es posible la exis-

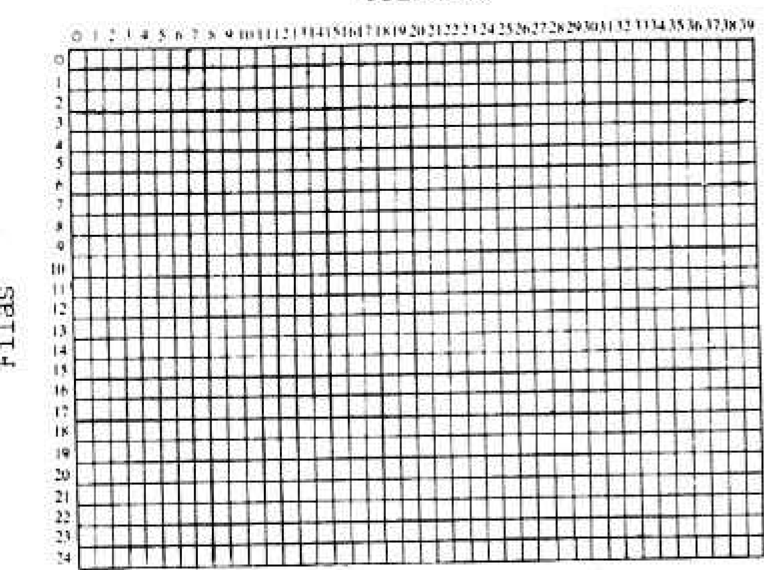


Fig. 1.13. La pantalla del C-64.

cio delante para poner el signo), indicándonos que es ahí donde comienzan la primera, segunda, tercera y cuarta zona de impresión, respectivamente. Así, el efecto de la coma es el de saltar al principio del siguiente cuarto de pantalla. Teclee, por ejemplo, el programa:

```
10 PRINT 1,2,3,4,5,6,7,8
20 PRINT "PEPE","JUAN","LUIS","MANUEL"
30 PRINT "NO VALE TANTO","PEPE"
```

Con las dos primeras líneas observaremos el comportamiento normal de la coma, pero en la línea 30 el primer campo ocupa más de 10 caracteres, por lo que, al saltar al principio de la zona siguiente, PEPE aparecerá impreso a partir de la columna 20. Pruebe a ver qué ocurre si tras el 8 de la línea 10 coloca una, dos, o tres comas. ¿Qué significa el resultado de ejecutar el programa?

Funciones de impresión

Puede que le parezca excesivo tanto truco de impresión, sin embargo, es algo que agradecerá cuando sus programas se vayan complicando y desee una buena presentación visual de la ejecución. Dos de los elementos más típicos en una buena impresión tienen que ver con el margen y con el espacio entre los elementos a imprimir.

Para resolver el problema del margen, el C-64 dispone una función llamada TAB (tabulación). Que ha de ir seguida de un número encerrado entre paréntesis. Este número indica en qué columna se va a imprimir el primer carácter de la expresión deseada (los números de columna, en la pantalla del Commodore van del 0 al 39). Por ejemplo, ensaye el programa siguiente (para borrar el programa anterior, teclee NEW y pulse RETURN):

```
10 PRINT TAB(11);"ESTO ESTA CENTRADO"
20 PRINT
30 PRINT
40 PRINT "ESTO NO"
50 PRINT TAB(5);"Y ESTO";TAB(30);"TAMPOCO"
```

En la primera línea utilizamos TAB(11) para centrar un literal. Recuerde que la pantalla tiene 40 columnas. Por otra parte, la frase ESTO ESTA CENTRADO ocupa 18 caracteres (acuérdese de contar los espacios en blanco). Así, 40-18=22, nos sobran 22 caracteres en la línea, y para centrar la frase tendremos que dejar 11 en el margen derecho y otros 11 en el izquierdo. En las líneas 20 y 30 se utilizan dos PRINT vacíos para provocar que la frase ESTO NO se escriba dos líneas más abajo de la frase anterior. El ordenador al leer un PRINT lo primero que hace es saltar a la línea siguiente. Si no hay nada para imprimir pasa a leer la instrucción siguiente. La línea 50 muestra cómo podemos utilizar diversos TAB en un PRINT. Tenga cuidado de no introducir un espacio en blanco entre el TAB y el primero de los paréntesis, pues obtendrá un mensaje de error. "?BAD SUBSCRIPT ERROR IN «núm. de línea» *x*", que indica que hay un error en el índice del TAB en la línea *x*, núm. de línea *x*.

> El separador «;» puede omitirse tras el TAB y también, en muchos casos, entre campos cuya separación esté bien definida. Por ejemplo, teclee:
>
> 100 PRINT "3+2=" 3+2
>
> y verá que no incurre en error por omitir el «;». Modifique la línea 50 del programa anterior, borrando todos los puntos y coma, y vuelva a ejecutarlo.

Borre el programa anterior NEW y RETURN, limpie la pantalla SHIFT y CLR/HOME y pasemos al siguiente ejemplo:

```
10 PRINT TAB(5)"DATO";TAB(20)"OTRO";TAB(
   30)"DATO"
20 PRINTSPC(5)"DATO";SPC(11)"OTRO";SPC(6
   )"DATO"
```

Ejecute el programa y observe que el efecto de ambas líneas es el mismo. En las líneas 10 utilizamos TAB para indicar la columna en que deseamos que se comience a escribir cada cadena, mientras que en la línea 20 empleamos SPC (abreviatura de SPACE, ESPACIO) para indicar el número de espacios en blanco que deseamos que existan antes de escribir cada cadena.

espera el ordenador comenzará la carga por su cuenta. Una vez realizada la carga con corrección nos informará con el mensaje:

LOADING
READY

Liste el programa y vuelva a utilizarlo, corrigiéndolo, ampliándolo, volviendo a hacer pruebas de grabación y reproducción, etc.

Existe otro método de cargar un programa de la Datassette. Si pulsa las teclas SHIFT y RUN/STOP, el ordenador cargará (LOAD) y ejecutará (RUN) automáticamente el primer programa que encuentre en la cinta.

Para cargar, y no ejecutar automáticamente, el primer programa que encuentre el ordenador en la cinta, teclee solamente:

LOAD

(sin ningún nombre de programa) y pulse RETURN.

Con esto ya está preparado para ir creando su propia biblioteca de programas. El trabajo de creación es pesado, y es aconsejable, para evitar la pérdida de tiempo, seguir los consejos siguientes:

— Realice copias de seguridad de los programas, en cintas distintas y consérvelas por separado.
— Mantenga las cintas lejos de las fuentes de calor, magnetismo o humedad; por ejemplo: radiadores, sol, televisores, etc. Es muy frecuente, al principio, dejar las cintas encima del televisor o monitor; evítelo.
— Con cada cinta, acostúmbrese a llevar una lista de los programas almacenados en ella y de «en qué vuelta» (utilizando el cuenta vueltas de la Datassette) comenzó la grabación de cada programa. Esto le ahorrará mucho tiempo de búsqueda.

Como resumen, recuerde que el formato general de las sentencias SAVE, LOAD, VERIFY es:

SAVE "«nombre de programa»"
LOAD "«nombre del programa»"
VERIFY "«nombre de programa»"

Nombre de programa: Cualquier secuencia de caracteres cuya longitud no exceda de 16. Podemos utilizar nombres más largos, pero el ordenador sólo utilizará los 16 primeros caracteres.

- Podemos prescindir de las comillas finales y el ordenador entenderá la instrucción.
- Podemos utilizar simplemente las palabras SAVE, LOAD o VERIFY y el Commodore:
  - Grabará (SAVE) el programa sin nombre.
  - Cargará (LOAD) el primer programa que encuentre.
  - Tratará de verificar (VERIFY) el primer programa que encuentre, con el que tiene en la memoria.

En este momento ya está provisto del material necesario para sacarle el máximo aprovechamiento a su aprendizaje del lenguaje BASIC en su C-64. Adelante y diviértase aprendiendo a controlar un ordenador.

2 Programación básica

INTRODUCCION: CONCEPTO DE VARIABLE

Utilizar PRINT para resolver cálculos aritméticos más o menos complicados puede resultar útil en múltiples ocasiones, pero, naturalmente, el gran auge y desarrollo de los ordenadores no se debe a que con ellos podamos realizar las mismas operaciones que con una calculadora de bolsillo.

Una de las características que hacen que los ordenadores sean tan versátiles y potentes es su capacidad para almacenar no sólo los programas, sino también los datos que éstos utilizan.

En el capítulo anterior vimos cómo almacenar un programa en la memoria del computador y, aún más, cómo grabarlo en un dispositivo accesorio para poder conservarlo en nuestra biblioteca de programas. Ahora hemos de aprender cómo almacenar y utilizar datos en el ordenador, y con ello viene una de las ideas más sencillas e importantes de la programación: el concepto de *VARIABLE*.

Digamos, en general, que una variable es un lugar de la memoria en el que podemos guardar el dato que queramos. Así, si deseamos almacenar el nombre de una serie de amigos o una serie de números, utilizaremos una serie de variables. El concepto de variable, *grosso modo*, es éste: el de un almacén.

Nombre de la variable →	A	B	C	....
Contenido de la variable →	2	4	—7	

Fig. 2.1. Ilustración del concepto de variable.

Para asimilar mejor este concepto suponga que la memoria está dividida en una serie de casillas (véase Fig. 2.1). En cada casilla podemos guardar un dato, un número o un nombre, y después utilizarlo cuando lo necesitemos. Mas para utilizar, o *llamar* a, un dato tendremos que disponer de alguna forma de reconocer en qué lugar está guardado. Así surge el concepto de *nombre de variable.* A una casilla le ponemos el nombre A y después introducimos en ella un dato. De este modo, refiriéndonos a la figura 2.1, en la casilla A tenemos un 2; es decir, la variable A tiene asignado un 2; la variable B contiene un 4, y en la variable C hemos introducido un —7. Acerca de los nombres que podemos utilizar para representar las variables, hablaremos en el cuarto apartado de este capítulo, pasando ahora a describir cómo podemos introducir contenidos en las variables y cómo podemos utilizarlas.

Asignación de contenido a variables numéricas

Pero, ¿cómo le decimos al ordenador que guarde un 2 en la variable A?, ¿cómo le decimos que una variable se va a llamar A? Para esto disponemos otro comando del BASIC, el comando LET. Para escribir este comando ter mos que atenernos a la sintaxis:

LET «nombre de variable» = «expresión»

Por ejemplo: Si introducimos LET A=2, al leer el computador esta orde lo primero que hace es buscar en su memoria si ya existe alguna variable q se llame A; si es así, borra el contenido anterior de la variable y le asigna valor 2. En caso contrario, toma una casilla vacía, le pone como nombre A introduce en ella el 2.

El número que se guarda en una variable puede ser resultado de una oper ción aritmética. Así, por ejemplo, LET B=2*3+1 significa que en la variab B se introducirá un 7. Considere el programa siguiente:

```
10 LET A=2
20 LET B=4
30 LET C=A+B
40 PRINT C
```

En la línea 10 el ordenador se encuentra con que tiene que tomar una ca lla, ponerle por nombre A e introducir en ella un 2. Asimismo, en la línea asignamos un 4 a la variable B. Al llegar a la línea 30 el ordenador interpre que tiene que definir una variable C e introducir en ella el resultado de sum los contenidos de las casillas A y B. Finalmente, en la línea 40 le ordenam al computador que busque el contenido de la casilla C y lo imprima en la pa talla. Por tanto, el resultado es que, al ejecutar el programa, aparecerá un 6 la pantalla. ¿Y tanto rollo para esto? Créalo, el salto es grande, antes sólo sab mos hacer cálculos con PRINT, ahora ya sabemos realizar operaciones sin r cesidad de imprimirlas en la pantalla, utilizar números y variables en operaciones y guardar los datos, los resultados intermedios y el valor final la memoria del computador. ¿Qué le parece el programa siguiente?:

```
10 REM ESTE PROGRAMA CALCULA LA LONGITUD Y EL
   AREA DE UN CIRCULO DE RADIO 5
20 LET R=5
30 LET L=2*π*R
40 LET A=π*R↑2
50 PRINT "LA LONGITUD DE LA CIRCUNFERENCIA
   DE R=";R;"ES";L;"Y SU AREA ES";A
```

Quizá lo más notable es la complicación del PRINT con gran cantidad campos numéricos y alfanuméricos separados por el símbolo de punto y con

Importante: *Recuerde que antes de comenzar a teclear un nuevo programa es aconsejable borrar el anterior (NEW y RETURN), pues si no puede entremezclar líneas de diferentes programas y obtener resultados no deseados.*

Como ya hemos mencionado, si en una variable que tuviera asignado un contenido introducimos otro, el contenido inicial se pierde. Compruébelo tecleando este programa:

```
10 LET A=5
20 LET A=10
30 PRINT"A=";A
```

Teclee RUN y pulse RETURN y verá que en la pantalla aparece A=10 ¿Y qué ha pasado con el 5? Muy sencillo, al introducir un 10 en A hemos borrado el 5 que contenía anteriormente.

Un punto a destacar es el significado del signo "=" que aparece en la sentencia de asignación. El signo igual de la sentencia LET no tiene nada que ver con la interpretación habitual en matemáticas. No significa que lo que aparece a la derecha del signo sea igual a lo que aparece a la izquierda, sino que en la variable que aparece a la izquierda del signo igual *hay que almacenar* el resultado de evaluar la expresión de la derecha. Considere el programa:

```
10 LET A=1
20 PRINTA
30 LET A=A+4
40 PRINTA
```

En la línea 10 la expresión A=1 asigna a la variable A el valor 1. Después, en la línea 30 nos encontramos con la expresión A=A+4. Es evidente que, matemáticamente hablando, esta expresión es incorrecta, pero es que en la sentencia LET esta expresión no significa que A sea igual a A+4, sino que *el nuevo valor de A será el resultado de sumar el antiguo valor de A (un 1) y el 4.* Al ejecutar el programa comprobará que en la pantalla aparecen los números 1 (línea 20) y 5 (línea 40).

La línea 10 del programa anterior realiza lo que en programación recibe el nombre de *inicialización de variables*. Si deseamos que una variable tenga cierto valor inicial, tenemos que dárselo. Si utilizamos una variable que no hemos inicializado, el Commodore le asigna directamente el valor inicial 0 Compruébelo borrando la línea 10 del programa anterior (teclee un 10 y pulse RETURN). Al ejecutar ahora el programa, en la pantalla aparecerán un 0 y un 4. Un 0 porque la línea 10 al tener que imprimir una variable no inicializada le asigna el valor 0, y un 4 porque en la línea 40 introducimos en A el resultado de sumar su antiguo valor, un cero, y un cuatro.

Trate de escribir un programa para intercambiar el contenido de dos variables. Esto es, queremos poner en A el número que hay en B, y en B, el que hay en A. No podemos hacerlo directamente, pues, por ejemplo, A=B introduce en A el valor de B, pero borra el valor inicial de A que después tendríamos que poner en B. Inténtelo y compare después su solución con el método propuesto en la figura 2.2 y en el programa siguiente.

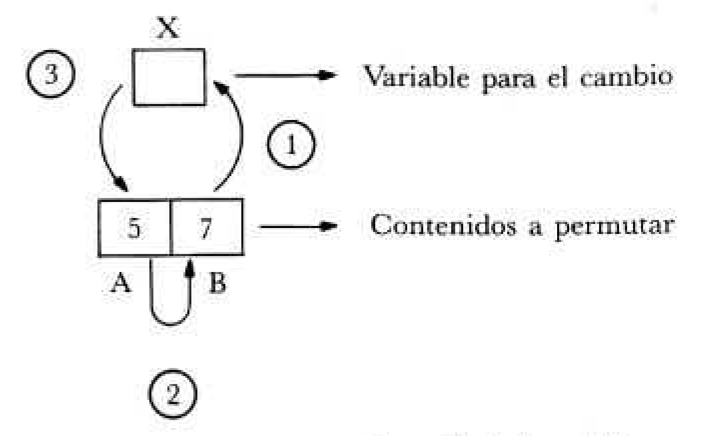


Fig. 2.2. Pasos para permutar el contenido de dos variables.

```
10 LET A=5
20 LET B=7
30 PRINT"ANTES DE LA PERMUTACION"
40 PRINT"A=";A
50 PRINT"B=";B
60 PRINT"X=";X
70 LET X=B
80 LET B=A
90 LET A=X
100 PRINT"DESPUES DE LA PERMUTACION"
110 PRINT"A=";A
120 PRINT"B=";B
130 PRINT"X=";X
```

Las líneas 10 a 60 se ocupan de inicializar las variables y de presentar sus valores en la pantalla. Como no hemos introducido ningún valor en la variable X, el C-64 le asigna el valor 0. El intercambio se realiza en las líneas 70 a 90, y las líneas 100 a 130 se dedican a la presentación del estado final en la pantalla.

Hasta ahora, para almacenar números, hemos utilizado un tipo de variables que podríamos llamar *variables numéricas normales* o *variables reales*. En estas variables podemos guardar datos tales como 5, 3.67, 412, 2.584 E09, etc. Sin embargo, si tenemos la seguridad de que lo que vamos a guardar son números enteros, podemos utilizar otro tipo de variable llamado variables enteras. Estas variables se caracterizan porque su nombre ha de ir seguido por el símbolo %, por ejemplo, A%, B%,... etc., para que el ordenador reconozca que deseamos que el contenido sea un número sin decimales. Ensaye el ejemplo siguiente:

```
10 REM UTILIZACION DE VARIABLES ENTERAS
20 LET A=6.3
30 LET B=5.2
40 LET A%=6.3
50 LET B%=5.2
60 PRINT"A=";A
70 PRINT"A%=";A%
80 PRINT"B=";B
90 PRINT"B%=";B%
100 PRINT"HE AQUI LA DIFERENCIA"
```

En las líneas 40 y 50, al tratar de introducir los números 6.3 y 5.2 en las variables enteras A% y B%, respectivamente, el ordenador toma sólo la parte entera del número, esto es, 6 y 5. ¡Tenga cuidado con esto! Por otra parte, fíjese en lo que ocurre en el programa siguiente:

```
10 REM COMPORTAMIENTO DE LAS VARIABLES
   ENTERAS
20 LET A%=6.3
30 LET B%=-7.3
40 PRINT"A%=";A%
50 PRINT"B%=";B%
```

Aquí comprobará que el ordenador realiza lo que en matemáticas recibe el nombre de *tomar la parte entera*, esto es, se toma el primer número entero cuyo valor sea menor que el número dado. El primer entero cuyo valor es menor que 6.3 es 6, pero el primer entero cuyo valor es menor que —7.3 es —8. Curioso, ¿no?

La importancia del empleo de variables enteras se basa, entre otras cosas, en que ocupan menos memoria (menos de la mitad que una varible real) y en que el ordenador trabaja más rápido con ellas. El ahorro de memoria irá tomando importancia según avance en la programación.

Asignación de contenido a variables alfanuméricas

Cuando introdujimos el comando PRINT vimos que el ordenador puede manejar dos tipos de datos: datos numéricos y datos alfanuméricos. Para almacenar datos numéricos disponemos de las variables numéricas, y, como usted ya se imaginaba, para almacenar datos alfanuméricos, nombres, direcciones, etc., existen *variables alfanuméricas*. En las variables alfanuméricas podemos guardar secuencias de caracteres, letras, dígitos, símbolos o caracteres gráficos. Para que el ordenador reconozca que estamos definiendo una variable alfanumérica tenemos que colocar el símbolo $ como final del nombre. Así, el programa:

```
10 LET A$="PILAR ES MI PRIMA"
20 PRINT A$
```

en la línea 10 guarda la cadena PILAR ES MI PRIMA en la variable A$ y después la línea 20 imprime en la pantalla el contenido de A$. Observe, en la línea 10, que el conjunto de caracteres que vamos a asignar a una variable alfanumérica ha de ir encerrado entre comillas.

> **Como siempre que hablamos de cadenas, las comillas finales pueden olvidarse, pero sólo si nos estamos refiriendo a una sentencia tan sencilla como la de la línea 10 del programa anterior. En otros casos se arriesga a obtener errores en la ejecución. Así que una buena norma es la de olvidarnos de esta particularidad del C-64 y teclear siempre las comillas finales.**

Un punto importante a recordar es que un espacio en blanco dentro de unas comillas es un carácter más. Un espacio en blanco fuera de las comillas es ignorado por el ordenador.

Operaciones con cadenas

Si bien podemos guardar dígitos en las variables alfanuméricas, no podemos realizar operaciones aritméticas con ellas, ya que cuando los números están en forma de cadena el ordenador los interpreta como caracteres y no como números. Así, la cadena "12" es la secuencia 1, 2, pero no el número 12. Compruebe que al ejecutar el programa:

```
10 LET A$="5"
20 PRINT A$*2
```

el ordenador le responde con el mensaje de error TYPE MISMATCH ERROR IN 20, que significa que en la línea 20 hay un error debido a tratar de operar con un tipo de dato equivocado.

Sin embargo, existen muchas formas de manejar las cadenas, y aunque muchas de ellas las veremos en el capítulo siguiente, vamos ahora a mencionar una operación alfanumérica llamada *concatenación*. El resultado de concatenar dos cadenas es otra cadena que contiene todos los caracteres de las dos primeras. El símbolo que se utiliza para representar la concatenación es el signo "+". Por ejemplo, el programa:

```
10 REM CONCATENACION
20 LET A$="HOLA"
30 LET B$=" PEPE"
40 LET C$=A$+B$
50 PRINT C$
```

imprime en la pantalla la cadena HOLA PEPE. Observe cómo en la línea 30 hemos comenzado la cadena con un carácter de espacio en blanco. Esto evita que en la cadena C$ las palabras HOLA y PEPE queden de la forma HOLAPEPE.

Vamos a fijarnos en la diferencia entre el signo + de adición y el signo + de concatenación. El programa siguiente trata de ilustrar este concepto.

```
10 REM DIFERENCIA ENTRE SUMA Y CONCATENACION
20 LET A=12.6
30 LET B=7.4
40 LET A$="12.6"
50 LET B$="7.4"
60 PRINT A+B
70 PRINT A$+B$
```

La línea 60 realiza e imprime la suma de los números 12.6 y 7.4, así en la pantalla aparece el número 20. Sin embargo, la línea 70 realiza e imprime la concatenación de las secuencias 12.6 y 7.4, y por ello en la pantalla aparece la secuencia 12.67.4. Claro, ¿no?

Hasta ahora hemos utilizado la estructura típica de la sentencia LET, sin embargo, muchos de los dialectos del BASIC, y entre ellos el del C-64, admiten la omisión de la palabra clave LET en la sentencia de asignación. De este modo, en el C-64, los dos programas siguientes son equivalentes.

```
10 REM UTILIZANDO LET
20 LET A$="PEPE"
30 LET B$=" VIVE EN "
40 LET C$="ARAGON"
50 LET A=5
60 LET B=6
70 PRINT A$+B$+C$
80 PRINT A;"*";B;"=";A*B
```

```
10 REM SIN UTILIZAR LET
20 A$="PEPE"
30 B$=" VIVE EN "
40 C$="ARAGON"
50 A=5
60 B=6
70 PRINT A$+B$+C$
80 PRINT A;"*";B;"=";A*B
```

Nombres y rangos de las variables

En general, el nombre de una variable responde a la siguiente estructura:

XYZ

Donde X: Ha de ser una letra, de la A a la Z.
Y: Puede ser una letra, de la A a la Z, o un dígito, del 0 al 9. Puede omitirse.
Z: — En variables numéricas reales no se tiene en cuenta.
— En variables numéricas enteras ha de ser símbolo, %.
— En variables alfanuméricas ha de ser el símbolo, $.

Nombres correctos
B
AL
A5
A%
A1%
AH%
B$
BS$
B4$

Nombres incorrectos	*Motivo*
B=	= No es letra ni dígito.
#	# No es letra ni dígito.
2B	Comienza con un dígito.
22	Comienza con un dígito.
A-%	— No es letra ni dígito.
1A%	Comienza con un dígito.
B,$	, No es letra ni dígito.
($)	( ) No es letra ni dígito.
1B$	Comienza con un dígito.

Fig. 2.3. Algunos nombres de variables correctos y otros incorrectos.

En la figura 2.3 se exponen algunos nombres correctos y otros incorrectos y, en este último caso, el motivo de su incorrección.

El nombre de una variable puede estar formado por más de dos caracteres, siempre y cuando el primero sea una letra, así al ejecutar:

```
10 REM NOMBRES MAS LARGOS
20 MARIO=5
30 ROSA=2
40 PRINT MARIO*ROSA
```

obtendrá un 10 en la pantalla. Incluso es posible darle un nombre de casi 80 caracteres a una variable. Lo más aconsejable es utilizar nombres de variables, con cuatro a seis caracteres, que nos recuerden para qué las empleamos, pero no utilizar nombres demasiado largos que nos cansaremos de teclear y que impedirán la legibilidad de nuestros programas.

Además hay que tener en cuenta que, aunque el nombre sea muy largo, el C-64 sólo reconoce los dos primeros caracteres para identificarla, y así los nombres:

SUMA
SUELDO
SUSANA
SUBMARINO

son interpretados, por el Commodore, como el mismo nombre de variable SU. Recuerde también que los espacios en blanco, dentro de un nombre de variable, son ignorados por el C-64. A continuación se exponen unos ejemplos de cómo se interpretan los nombres largos de variables:

Nombre de variable en el programa	*Representación interna en el C-64*
MARIO	MA
M ARIO	MA
NOMBES $	NO$
ENTERO %	EN%
ENTERO 1 %	EN%
H1233	H1

Y no sólo tendrá que cuidarse de incurrir en errores debido a esto, sino que además existe una serie de palabras que no podemos utilizar como nombres de variables, pues están reservadas por el Commodore para las palabras clave y para las funciones (ver Fig. 2.4).

ABS	DEF	GO TO	MID$	PRINT#	SPC	TI
AND	DIM	IF	NEW	READ	SQR	TIME
ASC	END	INPUT	NEXT	REM	ST	TI$
ATN	EXP	INPUT#	NOT	RESTORE	STATUS	TIME$
CHR$	FN	INT	ON	RETURN	STEP	TO
CLOSE	FOR	LEFT$	OPEN	RIGHT$	STOP	USR
CLR	FRE	LEN	OR	RND	STR$	VAL
CMD	GET	LET	PEEK	RUN	SYS	VERIFY
CONT	GET#	LIST	POKE	SAVE	TAB	WAIT
COS	GO	LOAD	POS	SGN	THEN	
DATA	GOSUB	LOG	PRINT	SIN	TAN	

Fig. 2.4. Palabras reservadas.

Estas palabras reservadas no pueden utilizarse como nombres de variables ni tampoco pueden aparecer dentro del nombre de una variable, y si acaso lo hacemos, el ordenador nos informará con un mensaje de SYNTAX ERROR. En la figura 2.5 se exponen algunos ejemplos.

Nombre de variable incorrecto	*Razón*
TOLEDO	Contiene la palabra TO
DIFERENCIA	Contiene la palabra IF
CONTADOR	Contiene la palabra CONT
SULTAN	Contiene la palabra TAN

Fig. 2.5. Ejemplos de nombres de variable que contienen palabras reservadas.

Otro punto a tener en cuenta cuando utilizamos variables es que el contenido que asignemos a una variable no exceda de los límites que esta variable puede manejar. ¿Y cuáles son estos límites? En cuanto a variables alfanuméricas, el número máximo de caracteres que les podemos asignar es de 255. Si sobrepasamos este límite obtendremos el mensaje de error: STRING TOO LONG ERROR (error por cadena demasiado larga). Claro que este límite sólo lo podremos sobrepasar concatenando cadenas, pues mediante un LET podemos introducir directamente menos de 80 caracteres (recuerde que la longitud máxima de una línea de programa es de 80 caracteres).

El contenido de las variables enteras tiene como límites, máximo y mínimo, los números +32767 y —32768, respectivamente. Números fuera de estos límites provocarán el mensaje de error: ILLEGAL QUANTITY ERROR. Compruébelo con las sentencias:

```
10 REM SOBREPASANDO LOS LIMITES DE UNA
   VARIABLE ENTERA
20 A%=50000
30 PRINTA%
```

Incluso para las variables reales existen unos límites respecto al tamaño de los números que pueden manejar. El límite inferior viene dado por la cantidad 2.93873588 E-39, tanto si lleva delante un signo positivo como si es negativo. El límite superior, tanto para números positivos como negativos, viene marcado por la cifra 1.70141183 E+38. De este modo, al ejecutar el programa,

```
10 REM LIMITES DE UNA VARIABLE REAL
20 A=2.93873587 E-39
30 PRINT A
40 A=-1.70141184 E+38
50 PRINT A
```

la línea 30 imprimirá un 0 en la pantalla, mientras que la línea 40 generará un mensaje de error: OVERFLOW ERROR IN 40 (error por número demasiado grande en la línea 40).

Y, para terminar con este recetario, un mensaje de error más. Si en un programa le surge el mensaje TYPE MISMATCH ERROR (error por tipo equi-

vocado de dato), puede que se deba a que ha tratado de introducir una cadena en una variable numérica. En una variable alfanumérica podemos introducir letras, dígitos, símbolos, etc., pero en una variable numérica, ya sea entera o real, sólo podemos introducir números.

Asignación de contenidos mediante el teclado: INPUT

La utilización de la sentencia de asignación LET, para dar valores a las variables, conduce a que si deseamos modificar los valores de las variables tengamos que cambiar las líneas del programa donde se realiza la asignación, y esto cada vez que queramos probar otros valores distintos. Para evitar esto utilizaremos la sentencia INPUT.

Fíjese en los programas:

```
10 REM PERIMETRO DE UN TRIANGULO
20 A=10
30 B=5
40 C=7
50 PRINT"PERIMETRO";A+B+C
```

```
10 REM PERIMETRO DE CUALQUIER TRIANGULO
20 INPUT A
30 INPUT B
40 INPUT C
50 PRINT "PERIMETRO=";A+B+C
```

En el primero asignamos valores a los lados A, B y C de un triángulo e imprimimos su perímetro. Si queremos calcular el perímetro de otro triángulo tendremos que modificar las líneas 20, 30 y 40. El segundo programa utiliza el comando INPUT. Al llegar el control de ejecución a la línea 20 aparecerá un símbolo de interrogación ? en la pantalla, con el que nos indica que está esperando a que tecleemos un número y pulsemos la tecla RETURN para que él lo lea. Lo mismo pasará en la línea 30 y después en la 40. Tras pulsar el último RETURN, el C-64 imprimirá en la pantalla el perímetro del triángulo. Con este programa podemos calcular tantos perímetros de triángulos como queramos y sin modificar ninguna línea. Cómodo, ¿no?

Pero imagínese que está escribiendo un programa para que lo utilice la secretaria de una gran empresa, y que ésta tuviera que introducir gran cantidad de información mediante los INPUT que usted ha puesto, por ejemplo, los nombres, la dirección y el teléfono de los clientes, o las compras o ventas de la semana. Suponga además que en medio de la introducción suena el teléfono o se va a tomar un café, ¿cómo se acuerda de en qué INPUT se quedó?, ¿en el lado B del triángulo?, ¿en el segundo apellido del quinto cliente? Para resolver esto se utilizan mensajes en el INPUT (en este sentido es bastante parecido al PRINT). Modifique el programa anterior para convertirlo en:

```
10 REM INPUT CON MENSAJES
20 INPUT "TECLEE EL PRIMER LADO";A
30 INPUT "TECLEE EL SEGUNDO LADO";B
40 INPUT "TECLEE EL TERCER LADO";C
50 PRINT"PERIMETRO=";A+B+C
```

Observe cómo los mensajes van separados de los nombres de variables mediante el símbolo de punto y coma. Ahora, cuando ejecutemos el programa, irán apareciendo mensajes, indicándonos qué tipo de respuesta se espera que tecleemos. Si nos equivocamos en la introducción y tecleamos una serie de letras, en lugar de un número, y pulsamos RETURN, el ordenador nos informará con el mensaje REDO FROM START, pero no se interrumpirá la ejecución, sino que volverá a presentarnos el INPUT en el que nos hemos equivocado, para darnos otra oportunidad.

Podemos utilizar un INPUT para introducir varios datos, por ejemplo:

```
10 REM INPUT MULTIPLE
20 INPUT "TECLEA LOS LADOS A,B Y C";A,B,C
30 PRINT "PERIMETRO=";A+B+C
```

En este caso podemos responder de dos formas:

a) Tecleando los tres números sucesivamente, separados por comas, y después pulsar RETURN.

b) Tecleando el primer número y pulsando RETURN, y así sucesivamente. (En este caso, cuando nos pida el segundo y tercer número, el C-64 imprimirá en pantalla dos signos de interrogación.)

Si estamos en el caso a) y en lugar de responder adecuadamente, por ejemplo, 2, 3, 4 RETURN, escribimos más respuestas de las necesarias, por ejemplo 2, 3, 4, 5, 6, 7, 8, RETURN, el ordenador nos sacará el mensaje EXTRA IGNORED y utilizará solamente el trozo 2, 3, 4 como respuesta al INPUT.

> - **Podemos utilizar un INPUT para pedir varios datos, pero si ponemos más de un mensaje en un INPUT, obtendremos un mensaje de error (SYNTAX ERROR).**
> - **Si en un INPUT múltiple nos equivocamos y tratamos de introducir un contenido alfanumérico en una variable numérica, obtendremos el mensaje REDO FROM START y el ordenador nos vuelve a pedir desde el primer dato del INPUT múltiple.**

También podemos utilizar el INPUT para asignar contenidos a variables alfanuméricas. Compare el funcionamiento de los dos programas siguientes:

```
10 REM DIRECCIONES
20 INPUT "COMO TE LLAMAS";A$
30 INPUT "DONDE VIVES";B$
40 PRINT A$;" VIVE EN ";B$
```

```
10 REM DIRECCIONES
20 PRINT "COMO TE LLAMAS";
30 INPUT A$
40 PRINT "DONDE VIVES ";
50 INPUT B$
60 PRINT A$;" VIVE EN ";B$
```

Aquí vemos que también podemos utilizar PRINT para escribir mensajes delante de los INPUT.

> **Sin embargo, hay que tener cuidado con la longitud de los mensajes, pues si superan la de una línea de la pantalla, el ordenador tomará como respuesta no sólo lo que usted haya tecleado, sino también el contenido del mensaje. Por otra parte, procure no teclear respuestas demasiado largas. Si su respuesta es más larga que la longitud de dos líneas de la pantalla, el ordenador sólo tomará la parte de la respuesta que exceda de las dos líneas de la pantalla.**

Como resumen, podemos decir que el formato general de la sentencia INPUT es:

> INPUT «lista de entrada»

> **Lista de entrada: Puede estar formada por diversos nombres de variables separados por comas. También puede llevar un mensaje delante de las variables (este mensaje ha de ir separado de las variables por punto y coma). Si tras la palabra INPUT no se coloca ni siquiera un nombre de variable, obtendrá el mensaje de error «SYNTAX ERROR».**

Si alguna vez, estando la ejecución en un INPUT, desea interrumpir el programa, pulse enérgicamente y a la vez las teclas RUN/STOP y RESTORE.

BUCLES Y CONDICIONES

Hasta ahora todos los programas que hemos visto pertenecen a los que llaman programas lineales. Cuando ejecutamos (RUN) el programa, el ordenador cede el control a la primera línea, cuando ésta ha finalizado cede el control a la segunda, y así sucesivamente hasta que ejecuta la última línea del programa. O sea que, de este modo, para cada acción individual necesitamos una línea de programa. Compare los dos programas siguientes:

```
10 REM PROGRAMA LINEAL
20 A$="ME LLAMO TARZAN"
30 PRINT A$
40 PRINT A$
50 PRINT A$
```

```
10 REM BUCLE INFINITO
20 A$="ME LLAMO TARZAN"
30 PRINT A$
40 GO TO 30
```

Con el primer programa necesitamos tres líneas para escribir tres veces la variable A$. Con el segundo veremos pasar por delante de nuestros ojos líneas y líneas de pantalla con la frase «ME LLAMO TARZAN», y esto lo hemos conseguido sólo con dos líneas. (Cuando se canse pulse la tecla RUN/STOP.) La razón se debe al comando GO TO 30 que aparece en la línea 40. GO TO 30 significa «vete a la línea 30». Así que cuando el control llegue a la línea 40 ésta cederá el control a la línea 30, que tras ejecutarse, pasará el control a la línea siguiente, la 40, y así sucesivamente. Quizá esto le parezca un poco inútil, pero hemos aprendido a desviar el control de la ejecución a la línea que queramos y a poder repetir una acción.

La estructura general de la sentencia GO TO es:

> GO TO «núm. de línea»

> **Núm. de línea: Este número ha de coincidir con el de alguna línea del programa, si no obtendremos el mensaje de error: UNDEF'D STATEMENT ERROR**

Sentencia IF...THEN

Claro que repetir una acción sin tener ninguna forma de controlar el número de veces que la vamos a repetir no es algo muy interesante. Para controlar esto, y muchas otras cosas, disponemos de la sentencia IF. Fíjese en el programa siguiente:

```
10 REM UTILIZACION DE IF
20 A=0
30 PRINT"LO VAMOS A ESCRIBIR DIEZ VECES"
40 PRINT
50 A=A+1
60 PRINT"ME LLAMO TARZAN"
70 IF A<10 THEN GO TO 50
80 PRINT
90 PRINT"YA ESTA"
```

Este programa imprimirá 10 veces en la pantalla la frase «ME LLAMO TARZAN». La línea 20 es innecesaria, pero no es malo que nos acustumbremos a inicializar las variables que vamos a utilizar en el programa. Después, en la línea 50 aumentamos el valor de A en una unidad, y en la línea 60 imprimimos por primera vez la frase deseada. La línea 70 compara el valor de A con el número 10. Si A (que es el número de veces que hemos escrito la frase «ME LLAMO TARZAN») es menor que 10, la línea 70 nos devuelve a la línea 50 para volver a imprimir la frase. Si A es 10 o más, ya hemos imprimido 10 veces la frase deseada y el control pasa a la línea 80 y después a la 90.

En este programa se introducen varios conceptos muy importantes. El primero es el concepto de contador. La variable A, en el ejemplo, es un contador,

pues con ella llevamos la cuenta del número de veces que imprimimos la frase. El segundo concepto es el de *bucle condicional,* si la comparación A < 10 es cierta volvemos a repetir el bucle. Si es falsa, el control pasa a la línea 80 y el programa finaliza en la línea 90.

La estructura de la sentencia IF es la más complicada y la más importante de las que vamos a ver. Muchas otras son variantes o adaptaciones de esta sentencia. Por ello es muy importante que domine esta estructura de control antes de pasar al apartado siguiente. El formato general de la sentencia IF es de la forma:

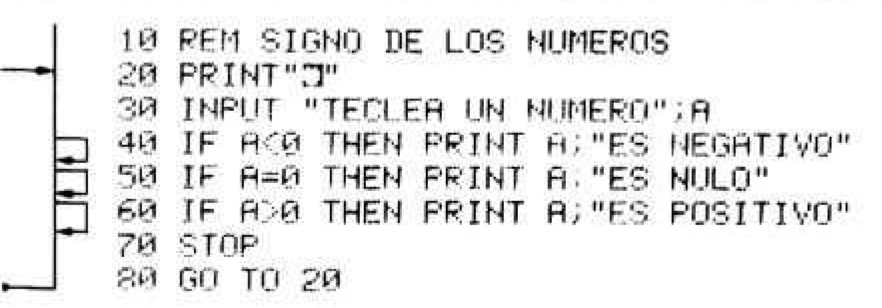
IF «expresión» «relación» «expresión» THEN «acciones».

Expresión: Representa cualquier constante o variable, ya sea numérica o alfanumérica, y cualquier expresión, ya sea numérica o alfanumérica.
Relación: Es cualquiera de las operaciones relacionales que se muestran en la figura 2.6.
Acciones: Son las órdenes que habrá que realizar en caso de que la comparación sea cierta. Por ejemplo, PRINT, o GO TO, o cualquier otro comando del BASIC, incluso otra sentencia IF.

Operador	*Significado*	*Prioridad*
=	Igual a	4
< > ó > <	No igual a (distinto de)	4
<	Menor que	4
>	Mayor que	4
< = ó = <	Menor o igual que	4
> = ó = >	Mayor o igual que	4

Fig. 2.6. Operadores relacionales.

Si la expresión condicional que hay dentro del IF («expresión» «relación» «expresión») es cierta, se realizan las acciones que hay detrás del comando THEN y después, a menos que estas acciones indiquen lo contrario (por ejemplo, mediante un GO TO), se pasa a la línea siguiente. Si la expresión condicional es falsa, se ignora el contenido del THEN y se pasa a la línea siguiente.



```
10 REM SIGNO DE LOS NUMEROS
20 PRINT"♥"
30 INPUT "TECLEA UN NUMERO";A
40 IF A<0 THEN PRINT A;"ES NEGATIVO"
50 IF A=0 THEN PRINT A;"ES NULO"
60 IF A>0 THEN PRINT A;"ES POSITIVO"
70 STOP
80 GO TO 20
```

En la línea 20 observará con extrañeza ese corazoncillo dentro del contenido del PRINT. Este símbolo representa la acción CLR, y cuando el ordenador lo encuentra dentro de un PRINT, limpia inmediatamente la pantalla. Para obtenerlo, tras pulsar las comillas, mantenga pulsada la tecla SHIFT y pulse una vez la tecla CLR. Fíjese en línea 40. Si el número que ha tecleado es negativo, la expresión relacional A < 0 será cierta y se pasa a realizar el contenido del comando THEN. Si es falsa, el control pasa a la línea siguiente. En la línea 70 nos encontramos con un nuevo tipo de instrucción, la sentencia STOP. Este comando provoca una parada temporal del programa y en la pantalla aparece el mensaje BREAK IN 70 (interrupción en la línea 70). Si introducimos la palabra clave CONT (CONTINUAR) el ordenador pasará el control a la línea siguiente a la que provocó la parada, esto es, a la línea 80, la cual envía el control a la 20, que se encarga de limpiar la pantalla, y el programa volverá a empezar. ¿Por qué hemos incluido la sentencia STOP? Si no lo hubiéramos incluido, la línea 80 cede el control a la 20 y ésta limpia la pantalla antes de que nos dé tiempo a leer el mensaje provocado por la línea 40, 50, ó 60. El formato general de los dos nuevos comandos es bastante sencillo.

STOP

CONT

Bueno, menos mal.

LÍNEAS MULTISENTENCIA

Considere el ejemplo siguiente:

```
10 REM ORDENACION DE NUMEROS
20 PRINT"♥"
30 INPUT "TECLEA UN NUMERO";X
40 INPUT "TECLEA OTRO NUMERO";Y
50 PRINT:PRINT:PRINT
60 IF X<Y THEN PRINT X;"ES MENOR QUE";Y
70 IF X=Y THEN PRINT X;"ES IGUAL QUE";Y
80 IF X>Y THEN PRINT X;"ES MAYOR QUE";Y
90 PRINT:PRINT:PRINT
100 PRINT "DESEA VOLVER A EMPEZAR. TECLEE"
110 INPUT "S O N Y PULSE RETURN"; A$
120 IF A$="S" THEN GOTO 20
130 IF A$<>"N" THEN PRINT"♥": GO TO 100
140 PRINT "SE ACABO"
150 END
```

En las líneas 50 y 90 nos encontramos con la primera novedad: varias órdenes en una misma línea. En este caso se debe a que queremos dejar tres líneas en blanco antes de volver a imprimir, y por ello introducimos tres comandos PRINT en una línea de programa. Observe que los tres comandos van separados por símbolos de dos puntos. Esto recibe el nombre de *línea multisentencia,* y en una línea podemos poner tantos comandos, separados por símbolos de dos puntos, como quepan en dos líneas de la pantalla (esto es, una línea de programa puede ocupar, como mucho, dos líneas de la pantalla).

Sin embargo, mediante un pequeño truco podemos introducir líneas de más de 80 caracteres.
Como puede verse en el apéndice de «Palabras clave», podemos teclear ciertas abreviaturas en lugar de las palabras clave. Así, en lugar de la palabra clave, PRINT podemos teclear un símbolo de interrogación ? Construya la línea de programa.
10 ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A: ?A
y pulse RETURN. Ahora introduzca el comando LIST y verá aparecer una línea de programa que ocupa más de cuatro líneas de pantalla, más de 160 caracteres. Curioso, ¿no?

Otro punto importante, del programa anterior, es cómo se utilizan las líneas 100 a 130 para preguntarle si desea volver a ejecutar el programa. Este tipo de construcción es extremadamente útil, así que es aconsejable que la estudie detenidamente. Entre las líneas 100 y 110 se le pregunta al usuario si desea volver a ejecutar el programa y se le indica que debe responder tecleando una S (para sí) o una N (para no) y pulsando RETURN. La línea 120 comprueba si ha pulsado la S y, en este caso, le devuelve al principio del programa. La línea 130 se encarga de controlar que no pulsemos respuestas distintas de S o N. Si acaso pulsó la N, el programa llega a la línea 150 donde nos encontramos con otro comando del BASIC, el comando END (FIN). La finalidad de esta palabra clave es bastante parecida a la del STOP. Esto es, cuando el programa llega a ella se para la ejecución y, en este caso, aparece el mensaje READY, indicando que ha llegado al final del programa. El comando END se utiliza para señalar, dentro del programa, en qué lugar termina éste, y es importante utilizarlo porque mejora la legibilidad de los programas y porque, como veremos, evita que surjan errores. Sin embargo, si teclea el comando CONT y existen más líneas tras la que contiene el END, la ejecución del programa continuará por la línea siguiente.

Es necesario volver a considerar cómo interpreta el ordenador la expresión relacional situada detrás del IF. Ya hemos dicho que si la expresión relacional es cierta, se pasa a ejecutar el contenido del THEN y se cede el control a la línea siguiente. Para el ordenador, una expresión relacional sólo puede tomar dos valores: cierto o falso. Si es cierta, el ordenador le asigna un -1, y si es falsa, le asigna un 0. Por ello, el resultado de la orden:

```
PRINT (2>3)  +  (1<2)  +  (2+3=5)
        0    +   -1    +   -1=-2
```

es un -2. La expresión (2>3) es falsa y, por tanto, le corresponde un 0. Las expresiones (1<2) y (2+3=5) son ciertas y por ello el ordenador les asigna el valor -1. Así, la suma de 0 + (-1) + (-1) da como resultado un -2. Esta evaluación que realiza el ordenador de las expresiones condicionales es muy útil en la programación. En el ejemplo siguiente se utiliza esta valoración para hallar el mayor de dos números.

```
10 REM MAYOR DE DOS NUMEROS
20 INPUT "TECLEE UN NUMERO";A
30 INPUT "TECLEE OTRO NUMERO";B
40 PRINT:PRINT:PRINT
50 PRINT TAB(7); "EL MAYOR ES";-A*(A>B)-
   B*(A<=B)
60 END
```

Si A>B la expresión −A*(A>B)−B*(A<=B) se convierte en −A*(−1)−B*(0)=A. Si A<=B, la expresión −A*(A>B)−B*(A<=B) se convierte en −A*(0)−B*(−1)=B

Este concepto es mucho más útil de lo que pueda imaginar ahora. Pero mientras tanto vamos a ver una aplicación que en principio parece más interesante. Vamos a diseñar un programa que sirva para calcular la nota media de una clase:

```
10 REM NOTA MEDIA
20 INPUT"CUANTOS ALUMNOS";N
30 C=0: SUMA=0
40 C=C+1
50 PRINT "TECLEE LA NOTA DEL";C; "ALUMNO";
60 INPUT NO
70 SUMA=SUMA+NO
80 PRINT"J"
90 IF C<N THEN 40
100 PRINT "LA NOTA MEDIA DE LOS";N;
    "ALUMNOS ES";SUMA/N
110 END
```

Este programa sirve para hallar la nota media de cualquier clase, ya que el número de alumnos, N, lo tenemos que introducir en la línea 20. Otro concepto importante a recordar es el concepto de contador. Utilizamos como contador a la variable C y ésta va aumentando de 1 en 1 (línea 40) hasta que llega al tamaño de la clase, N (línea 90). Con este programa puede hallar la nota media de una clase, ya tenga ésta 20, 40, 100 ó 1000 alumnos. Fíjese, además, en la utilización de la línea 50 para que nos indique el número del alumno cuya nota tenemos que introducir. Adicionalmente, en la línea 90 observamos cómo la sentencia THEN GO TO 40 puede abreviarse con la sentencia THEN 40.

La sentencia THEN GO TO «núm. de línea», puede abreviarse mediante:

THEN «núm. de línea»

o mediante:

GO TO «núm. de línea»

El comando GET

Este comando es otro de los utilizados, al igual que INPUT, para introducir datos desde el teclado. Sin embargo, su funcionamiento es bastante distinto.

Cuando el control llega al GET, éste revisa el teclado, y si pulsamos alguna tecla, trata de introducir el carácter correspondiente en la variable que va a continuación del GET. El formato general de esta palabra clave es:

GET «nombre de variable»

Nombre de variable. Puede ser:
a) Variable alfanumérica, por ejemplo, N$.
b) Variable real, por ejemplo, N.
c) Variable entera, por ejemplo, N%.

Por ejemplo, al ejecutar el programa:

```
10 REM CARACTERES CORRIENDO
20 GET N$
30 PRINTN$
40 GOTO 20
```

Cuando el control llega a la línea 20, el comando GET hace una revisión del teclado e introduce la tecla que hayamos pulsado en N$. Después, la línea 30 imprime el contenido de N$ y la línea 40 devuelve el control a la línea 20, la cual vuelve a realizar una revisión del teclado, etc. Observe que toda esta operación se realizará casi instantáneamente y, por tanto, es bastante difícil que usted sea capaz de pulsar una tecla cada vez que se realiza una revisión del teclado. Por ello, la mayoría de las veces, cuando se ejecute el GET, usted no habrá pulsado tecla alguna y GET introducirá un vacío en N$ y después la línea 30 imprimirá un vacío (es decir nada) en la pantalla. Por esto es por lo que usted ve cómo los caracteres que ha pulsado van corriendo hacia el borde superior de la pantalla. Para evitar esto podemos modificar el programa anterior en la forma (para parar el programa pulse la tecla RUN/STOP):

```
10 REM DETECTA CARACTERES
20 GET N$:IF N$="" GOTO 20
30 PRINT N$
40 GO TO 20
```

La modificación en la línea 20 nos asegura que el control NO pasará de esta línea, a menos que pulsemos una tecla de carácter. Con este programa aparecerá un carácter en cada línea de pantalla. Si quiere escribir en una misma línea, basta con añadir un punto y coma al final de la línea 30. Escriba de esta forma y después estudie qué es lo que pasa si pulsa las teclas de acción DEL, CLR, CRSR, etc.

El nombre de variable que aparece en el GET puede ser el de una variable numérica real o una entera, por ejemplo, GET N o GET N%. Pero en este caso sólo aceptará que pulsemos teclas numéricas. En cuanto pulsemos una tecla que no sea numérica, aparecerá el mensaje SYNTAX ERROR.

Los operadores de Boole

Podemos sofisticar aún más las expresiones relacionales que acabamos de ver utilizando los llamados operadores de Boole. Estos se representan mediante las palabras NOT (no), AND (y) OR (o), y sus niveles de prioridad se expresan en la figura 2.7.

Operador	*Prioridad*
NOT	3
AND	2
OR	1

Fig. 2.7. Operadores de Boole

Bueno, y ¿cómo se utiliza esto? Quizá lo mejor es estudiar las llamadas «tablas de verdad». Suponga que tenemos dos expresiones condicionales A y B (observe que A y B serán expresiones del tipo «NUMERO<5»; «X+Y<>7», A$=B$, etc.) Para concretar, imaginemos que A es el suceso «Tengo coche» (puede ser cierta o falsa), que B es el suceso «Tengo dinero», y supongamos un tercer suceso C= «Me voy a la sierra». Así la expresión:

IF A THEN C

significa: «*Si* tengo coche, *entonces* me voy a la sierra», y la expresión:

IF B THEN C

quiere decir: «*Si* tengo dinero, *entonces* me voy a la sierra.»

Considere ahora las siguientes sentencias compuestas:

IF A AND B THEN C

Esta sentencia la podríamos traducir por: «*Si* tengo coche *y* tengo dinero, entonces me voy a la sierra.» Esto indica que me voy a la sierra sólo si tengo coche y si tengo dinero, es decir, para poder hacer C han de ser ciertas A y B. Sin embargo, la sentencia:

IF A OR B THEN C

indica que: «*Si* tengo coche *o bien* tengo dinero, *entonces* me voy a la sierra.» Esto es, en este caso, me voy a la sierra si bien tengo coche o si sólo tengo dinero, y si se cumplen las dos, mucho mejor, con mayor motivo me voy a la sierra.

Resumiendo, el conectivo AND implica que para poder realizar C han de cumplirse tanto A como B, mientras que con el conectivo OR basta con que se cumpla o bien A, o bien B, o ambas. Esto se resume en las siguientes tablas de verdad.

A	*B*	*A OR B*
Cierta	Cierta	Cierta
Cierta	Falsa	Cierta
Falsa	Cierta	Cierta
Falsa	Falsa	Falsa

A	*B*	*A AND B*
Cierta	Cierta	Cierta
Cierta	Falsa	Falsa
Falsa	Cierta	Falsa
Falsa	Falsa	Falsa

Fig.2.8. Tablas de verdad de los conectivos AND y OR.

Más sencilla de explicar y, generalmente, más difícil de observar dónde se ha de utilizar es la partícula NOT. Suponga que el suceso A es «Hay patatas», y C el suceso «Compro dos kilos». Así, la sentencia:

IF A THEN C

indica: «*Si* hay patatas, *entonces* compro dos kilos.» Imagínese ahora un suceso D= «Me voy a otra tienda», y de este modo podemos escribir:

IF NOT (A) THEN D

que significa: «*Si* no hay patatas, entonces me voy a otra tienda.» Fácil, ¿no? Es decir, NOT (A) es el suceso contrario de A. Claro, es difícil ver cuándo nos interesa más escribir, por ejemplo, «X < >3» ó NOT (X=3), pues ambas expresiones reflejan lo mismo, «X es distinto de 3», o «No se cumple que X sea igual a 3». La tabla de verdad de la partícula NOT se expresa en la siguiente figura:

A	*NOT (A)*
Cierta	Falsa
Falsa	Cierta

Fig. 2.9 Tabla de verdad de la partícula NOT.

Ahora, lo mejor es refrescarnos un poco y repasar las ideas tecleando el ejemplo siguiente:

```
1Ø REM CALIFICACION
2Ø PRINT"♫"
3Ø INPUT"NOMBRE DEL ALUMNO";N$
4Ø PRINT:PRINT"NOTA DE ";N$;
5Ø INPUT N
6Ø IF N<Ø OR N>1Ø THEN PRINT"♫":GO TO 4Ø
7Ø PRINT:PRINT:PRINT
8Ø IF NOT(N<5) THEN 3ØØ
9Ø PRINT N$;" NO HA SUPERADO EL EXAMEN"
1ØØ PRINT "EN SEPTIEMBRE LE IRA MEJOR"
11Ø PRINT:PRINT:PRINT
12Ø PRINT "DESEA VOLVER A EMPEZAR"
13Ø PRINT"PULSE LA N O LA S"
14Ø GET R$:IF R$="" THEN 14Ø
15Ø IF R$="S" THEN 2Ø
16Ø IF NOT(R$="N") THEN 14Ø
17Ø PRINT"HASTA OTRA"
18Ø END
3ØØ PRINT N$;" HA SUPERADO EL EXAMEN. SU
    CALIFICACION HA SIDO DE ";
31Ø IF(N>=5)AND(N<7)THEN PRINT"APROBADO"
32Ø IF(N>=7)AND(N<9)THEN PRINT"NOTABLE"
33Ø IF(N>=9)AND(N<1Ø)THEN PRINT
    "SOBRESALIENTE"
34Ø IF N=1Ø THEN PRINT"MATRICULA DE HONOR"
35Ø GO TO 11Ø
```

De acuerdo, de acuerdo, la construcción es algo artificial, pero es que es difícil reunir las tres partículas en un ejemplo tan sencillo.Es obvio que en lugar de la línea 8Ø podríamos haber puesto:

8Ø IF N>=5 THEN 3ØØ

Pues ambas indican lo mismo. O que en lugar de la línea 15Ø podríamos haber escrito:

15Ø IF R$ < > "N" THEN 14Ø

Pero esto nos ilustra que existen muchas formas de escribir lo mismo. De ahora en adelante lo que hemos de intentar es escribir del modo más lógico y simple.

Otros dos puntos destacables en el programa anterior es cómo utilizamos la línea 6Ø para tratar de evitar que el usuario introduzca una nota fuera del rango de Ø a 1Ø y cómo las líneas 15Ø y 16Ø se encargan de que sólo podamos responder con una S o con una N.

La sentencia ON... GO TO

En muchas ocasiones utilizamos la sentencia IF... THEN para comprobar si una variable tiene cierto valor, y en ese caso enviamos el control a cierto número de línea (GO TO núm. de línea). Por ejemplo, considere el programa:

```
1Ø REM RAMIFICACION
2Ø PRINT "PULSE UNA TECLA DEL 1 AL 5"
3Ø GET N%: IF N%=Ø THEN 3Ø
4Ø PRINT "♫"
5Ø IF N%=1 THEN 1ØØ
6Ø IF N%=2 THEN 15Ø
7Ø IF N%=3 THEN 2ØØ
8Ø IF N%=4 THEN 25Ø
85 IF N%=5 THEN 3ØØ
9Ø GO TO 2Ø
1ØØ PRINT "AGORERO": GO TO 2Ø
```

```
150 PRINT "PUSILANIME": GO TO 20
200 PRINT "ESNORTAO": GO TO 20
250 PRINT "MALAJE": GO TO 20
300 PRINT "REQUETESALAO": GO TO 20
```

De este modo, el mensaje que obtengamos dependerá de la tecla numérica que pulsemos. Si pulsamos una tecla numérica distinta del 1, 2, 3, 4 ó 5, la línea 90 nos devolverá a la línea 20. Si pulsamos una tecla no numérica obtendremos un mensaje de error. Observe la estructura de la línea 30. La comparación N% = 0 se debe a que cuando el GET es numérico y no pulsamos tecla alguna, el ordenador introduce un 0 en N%. Con esta comparación aseguramos que mientras no pulsemos una tecla el control no pasará de la línea 30. (Si pulsásemos la tecla 0 introducimos también un 0 en N%.) Pues bien, este tipo de ramificación (que se dirige hacia un número de línea según el número que hemos pulsado) es tan típica que el C-64 incorpora un tipo de estructura particular para este caso: la sentencia ON...GOTO. Podemos modificar el programa anterior en la forma:

```
10 REM RAMIFICACION
20 PRINT "PULSE UNA TECLA DEL 1 AL 5"
30 GET N%: IF N%=0 THEN 30
40 PRINT "♥"
50 ON N% GOTO 100,150,200,250,300
60 GO TO 20
100 PRINT "AGORERO": GO TO 20
110 PRINT "PUSILANIME": GO TO 20
150 PRINT "PUSILANIME": GO TO 20
200 PRINT "ESNORTAO": GO TO 20
250 PRINT "MALAJE": GO TO 20
300 PRINT "REQUETESALAO": GO TO 20
```

Aquí, la línea 50 requiere la mayor atención. Esta línea suple a las líneas 50, 60, 70, 80 y 90 del programa anterior. El significado es que si N% vale 1, 2, 3, 4 ó 5 el control se cederá a las líneas 100, 150, 200, 250 ó 300, respectivamente. Así, la estructura general es:

ON «expresión» GOTO «1er núm. de línea, 2° núm. de línea...»

Expresión: Tal como A o A+4, que el ordenador evalúa y toma la parte entera del resultado. Si el resultado es 1, se cede el control al primer número de línea de la que hay tras él GOTO. Si el resultado es 2, al segundo número de línea, y así sucesivamente. Si hay cinco números de línea tras el GOTO, entonces expresión puede tomar valores del 1 al 5. Un valor de expresión fuera de este rango ocasionará que el control pase a la línea siguiente a la de la sentencia ON...GOTO. Aunque es correcto escribir el comando GOTO con un espacio en blanco intermedio, esto no se admite cuando forma parte de la sentencia ON...GOTO, y el C-64 le avisará mostrándole un mensaje de SYNTAX ERROR.

Antes de continuar es conveniente realizar una revisión de los conceptos. Si se trata de conceptos de programación, no hay nada mejor que intentar construir programas, así que ¡adelante! Antes de leer la solución propuesta trate de obtenerla, personalmente, con la seguridad de que aprenderá mucho más así que revisando el programa listado en el libro.

En primer lugar, imagine que vamos a realizar la elección de «MISS COMMODORE». El ordenador nos irá preguntando por el nombre, talla y peso de las candidatas, y nos indicará si han sido admitidas o rechazadas. Claro que el ordenador no tiene criterios para relacionar, así que le indicaremos que han de medir entre 1.65 y 1.80 metros y han de pesar entre 45 y 60 kg.

```
10 REM MISS COMMODORE
11 REM
12 REM
15 PRINT "♥"
20 PRINT:PRINT:INPUT"NOMBRE";N$
30 PRINT:PRINT:INPUT"TALLA";T
40 PRINT:PRINT:INPUT"PESO";P
50 IF T<1.65 OR T>1.80 OR P>60 OR P<45
   THEN PRINT "♥QQQQRECHAZADA";:GOTO 70
60 PRINT "♥QQQQADMITIDA";
70 PRINT " LA CANDIDATA ";N$
100 PRINT:PRINT:PRINT:PRINT
    "OTRA CANDIDATA (RESPONDA SI O NO)";
110 INPUT C$
115 IF C$<>"SI" AND C$<>"NO" THEN
    PRINT"QQQRESPONDA SI O NO":GOTO100
120 IF C$="SI" THEN15
130 PRINT "♥QQQQQQQQ]]]]]]]]]]]]]FIN DE"+
    " LA SELECCION"
140 END
```

Fíjese en los símbolos que aparecen dentro de los PRINT de las líneas 15, 50, 60, 115 y 130. Hay tres tipos de símbolos: el corazón [♥], del que ya hemos hablado; una especie de «q» [Q], y un símbolo de corchete [ ]]. El [♥] representa la acción CLR; la [Q] es CRSR↓, y el [ ]] es CRSR→. Cuando pulsamos las teclas correspondientes, dentro del símbolo de comillas (ver el capítulo 1), aparecen estos símbolos, y cuando el ordenador va a ejecutar el PRINT, realiza la acción que le indica el símbolo correspondiente. Con ello podemos escribir en el lugar que queramos de la pantalla y presentar nuestros resultados de una forma clara y ordenada.

El resto del programa no aporta mucho más, a menos que usted se decida a incluir en la selección sus preferencias personales (por ejemplo que sea rubia, ojos verdes, etc.). ¿Qué le parece obligar al ordenador a que acepte a la candidata COMMODORE sea como sea (es la hija del jefe)?

El programa siguiente puede ser una fuente para que usted desarrolle programas educativos para jugar con sus amigos o hijos. Esta sencilla estructura

inicial sirve para repasar una tabla de multiplicar. El C-64 le pregunta qué número desea repasar, línea 40, y tras aceptar su respuesta, por ejemplo, el 8, empieza a plantearle preguntas en la forma:

8 * 1=?

Ahora ha de teclear su respuesta, en este caso 8, y pulsar RETURN. El Commodore le indicará si ha acertado o fallado y le mostrará la respuesta correcta. Si el número de aciertos es mayor o igual que 7, podrá pasar a otra tabla, si no tendrá que repasarla de nuevo.

```
10 REM TABLAS DE MULTIPLICAR
11 REM
12 REM
20 REM A=ACIERTOS
30 PRINT "♥"
40 INPUT "QUE NUMERO DESEA REPASAR";N
50 C=1:A=0
60 PRINT:PRINT:PRINT
70 PRINT N;"*";C;"=";
80 INPUT R
90 IF R=N*C THEN PRINT TAB(9);"BIEN";:A=A+1
100 IF R<>N*C THEN PRINT TAB(9);"MAL";
110 PRINT N;"*";C;"=";N*C
120 C=C+1
130 IF C<=10 THEN 60
140 IF A<7 THEN PRINT "MUY MAL. TIENE"+
    " QUE VOLVER A REPASARLA":GOTO 50
150 PRINT "BIEN, DESEA REPASAR OTRA"+
    " TABLA(S/N)"
160 GET R$:IF R$<>"S" AND R$<>"N"
    THEN 160
170 IF R$="S" THEN 30
180 PRINT "♥QQQQQQQQQQQ"
190 PRINT TAB(15);"HASTA OTRA"
200 END
```

Intente introducir la siguiente variante: si se fallan dos respuestas consecutivas, el ordenador nos obliga, inmediatamente, a repasar la tabla.

Como ejercicio final, suponga que el ordenador es una tienda de comestibles y que usted es el cliente que va de compras. En el almacén hay una serie de productos, por ejemplo, macarrones, spaghettis y tallarines. El vendedor tiene que saber en todo momento cuál es el precio de sus productos y qué cantidad de cada uno de ellos tiene en el almacén. Nos preguntará sucesivamente qué productos deseamos y tras comprobar que tiene suficiente para servirnos nos pasará la cuenta final. Procure que nadie pueda engañar al dependiente.

```
10 REM DIA DE COMPRAS
20 REM PM%,PS%,PT%, SON LOS PRECIOS
30 REM TM,TS,TT, SON LAS EXISTENCIAS
40 REM CM,CS,CT, SON LAS CANTIDADES
   COMPRADAS
50 PM%=90:PS%=150:PT%=120
60 TM=24.5:TS=37.5:TT=20
88 REM
89 REM
90 REM PROGRAMA PRINCIPAL
91 REM
92 REM
100 PRINT"♥QQQQQQQQQQQQ"
110 PRINT TAB(12);"DIA DE COMPRAS"
120 PRINT: PRINT:PRINT
130 PRINT "PARA COMPRAR PULSE LA TECLA C"
140 GET A$: IF A$<>"C" THEN 140
150 PRINT "♥QQQQQQQQQ"
160 PRINT TAB(10);"?QUE DESEA COMPRAR?"
170 PRINT:PRINT TAB(14);"1.MACARRONES"
180 PRINT:PRINT TAB(14);"2.SPAGUETIS"
190 PRINT:PRINT TAB(14);"3.TALLARINES"
200 PRINT:PRINT:PRINT
210 PRINT"PULSE EL NUMERO DEL PRODUCTO"+
    " DESEADO"
220 GET A%: IF A%<>1 AND A%<>2 AND A%<>3
    THEN 220
230 PRINT "♥QQQQQQQQQQQ"
240 ON A% GOTO 1000,1500,2000
250 PRINT "♥QQQQQ"
260 FACTURA=PM%*CM+PS%*CS+PT%*CT
270 PRINT "SU FACTURA ES DE";FACTURA;
    "PTAS"
280 PRINT:PRINT:PRINT
285 PRINT TAB(6);"DESEA ALGO MAS S/N?"
290 GET R$: IF R$<>"S" AND R$<>"N"
    THEN 290
300 IF R$="S" THEN 150
310 PRINT:PRINT:PRINT
315 PRINT "CON CUANTO DINERO ME PAGA";
320 INPUT DINERO
340 IF DINERO<0 THEN PRINT
    "NO EMPECEMOS CON BROMAS":GOTO 310
350 IF DINERO < FACTURA THEN PRINT
    "NO ES SUFICIENTE":GOTO 310
360 VUELTA = DINERO-FACTURA
370 PRINT:PRINT "TOME LE SOBRAN";VUELTA;
    "PTAS."
```

```
380 PRINT:PRINT:PRINT TAB(15);
    "HASTA OTRA"
390 END
998 REM
999 REM
1000 REM VENTA DE MACARRONES
1001 REM
1002 REM
1010 PRINT "CUANTOS KILOS DESEA";
1020 INPUT CANTIDAD
1030 IF CANTIDAD<0 THEN 1010
1040 IF CANTIDAD>TM THEN PRINT
     "SOLO TENGO";TM;"KILOS":GOTO 1010
1050 CM=CM+CANTIDAD:TM=TM-CANTIDAD
1060 GOTO 250
1498 REM
1499 REM
1500 REM VENTA DE SPAGUETIS
1501 REM
1502 REM
1510 PRINT "CUANTOS KILOS DESEA";
1520 INPUT CANTIDAD
1530 IF CANTIDAD<0 THEN 1510
1540 IF CANTIDAD>TS THEN PRINT
     "SOLO TENGO";TS;"KILOS":GOTO 1510
1550 CS=CS+CANTIDAD:TS=TS-CANTIDAD
1560 GOTO 250
1998 REM
1999 REM
2000 REM VENTA DE TALLARINES
2001 REM
2002 REM
2010 PRINT "CUANTOS KILOS DESEA";
2020 INPUT CANTIDAD
2030 IF CANTIDAD<0 THEN 2010
2040 IF CANTIDAD>TT THEN PRINT
     "SOLO TENGO";TT;"KILOS":GOTO 2010
2050 CT=CT+CANTIDAD:TT=TT-CANTIDAD
2060 GOTO 250
```

Las líneas 50 y 60 establecen los precios y existencias de los macarrones, spaghettis y tallarines. Después, la línea 140 nos exige que pulsemos la tecla C para entrar en la tienda, y las líneas 170, 180 y 190 nos muestran los productos en venta. La línea 220 se encarga de que tengamos que pulsar el 1, el 2 ó el 3 para poder comprar (evidentemente, si pulsa una tecla de letra se detendrá la ejecución del programa y obtendrá un mensaje de error, pero ya veremos cómo evitar esto en el capítulo siguiente). La línea 240 nos envía a la 1000, 1500 ó 2000, según el producto que deseemos comprar. Observe que la estructura de los grupos de líneas 1000-1060, 1500-1560, y 2000-2060 es similar y evita que podamos comprar cantidades negativas o superiores a las existencias (1030, 1040), y después calcula las cantidades totales compradas y el nuevo valor de las existencias. Tras esto, las líneas 1060, 1560 y 2060 devuelven el control a la línea 250 y las líneas 260-300 se encargan de presentarnos la factura acumulada y de preguntarnos si todavía deseamos algo más. Si no es así, el control pasa a la línea 310, que nos pide que paguemos la factura, y después las líneas 320-390 se encargan de realizar el cobro, sin permitir que le engañemos, de darnos la vuelta y de finalizar con una amable despedida.

Pero ésta es una tienda muy poco surtida. Aproveche la estructura de las líneas 1000-1060, 1500-1560 y 2000-2060 e introduzca nuevos productos. Acuérdese de adecuar las líneas 50, 60, 220, 240, 260 y de introducir las nuevas líneas necesarias.

El bucle FOR...TO...NEXT

Al comenzar con la sentencia IF...THEN utilizábamos un ejemplo con el que imprimíamos 10 veces una frase en la pantalla. Revíselo un momento. Para ello utilizábamos una variable (contador) con la que llevábamos la cuenta del número de veces que habíamos impreso la frase. Tras cada impresión, comparábamos si ya lo habíamos hecho 10 veces. Si así era, terminaba el programa, pero en caso contrario aumentábamos el contador en una unidad y volvíamos a imprimir la frase y aumentar el contador, y así hasta 10 veces.

Este tipo de acción, repetir un número de veces ciertas cosas, es tan frecuente que se ha diseñado una sentencia especial para utilizarla en estos casos: la sentencia FOR...TO...NEXT. Compare los dos programas siguientes:

```
10 REM COMPARACION
20 C=1
30 PRINT"COMMODORE"
40 C=C+1
50 IF NOT(C>10) THEN 30
60 PRINT"SE ACABO,C=";C
```

```
10 REM COMPARACION
20 FOR C=1 TO 10
30 PRINT"COMMODORE"
40 NEXT C
50 PRINT"SE ACABO,C=";C
```

La similitud no es tan evidente, y por ello es aconsejable que vaya fijándose en la estructura del primer programa mientras que vamos hablando del segundo. En la línea 20 nos encontramos con la sentencia FOR C = 1 TO 10 (desde que C vale 1 hasta que valga 10), que lo primero que hace es introducir en C el valor 1 y ceder el control a la línea siguiente. En la línea 30 imprimimos la palabra «COMMODORE» y en la línea siguiente NEXTC aumenta en 1 el valor de C y devuelve el control a la línea 20. Ahora C vale 2 y lo primero que hace la línea 20 es comparar si C es mayor que 10. Como esto no es cierto, pues C vale 2, el control pasa a la línea 30 y después la 40, que incrementa en 1 el valor de C, y así sucesivamente. Cuando la línea 40 introduce en C el

valor 1Ø y cede el control a la 2Ø, todavía se vuelve a repetir el bucle, pues 1Ø no es mayor que 1Ø. Sin embargo, tras esta repetición, el valor de C pasa a ser 11 y entonces la comparamos a la línea 2Ø. «Es C mayor que 1Ø» es cierta y el control pasa directamente a la línea 5Ø.

Así, la sentencia NEXT C viene a ser como la sentencia C=C+1 del primer programa, y después cede el control a la línea FOR...TO. La sentencia FOR C=1 y la comparación de C con el valor máximo permitido 1Ø. En cuanto el valor de C sobrepase el límite 1Ø ya no se volverá a realizar el bucle.

ESTRUCTURA GENERAL DE LA SENTENCIA FOR

El formato de la sentencia FOR es:

FOR «nombre de variable»=«valor inicial» TO «valor final» STEP «incremento»

....
.... (Instrucciones a repetir)
....

NEXT «nombre de variable»

Nombre de variable: Es cualquier nombre de variable real, es decir, no podemos utilizar nombres de variables enteras ni alfanuméricas.
Valor inicial: igual que incremento.
Valor final: igual que incremento.
Incremento: Cualquier expresión aritmética. Pero recuerde que éste tomará el valor entero del resultado y que sólo se evalúa al principio del bucle.

El nombre de variable, llamado variable índice, es cualquier nombre de variable real (recuerde que el C-64 sólo tiene en cuenta los dos primeros caracteres) y hemos de poner el mismo nombre tras el NEXT. Así, de los programas:

```
10 FOR I=1 TO 10
20 PRINT I
30 NEXT I
```

```
10 FOR I=1 TO 10
20 PRINT I
30 NEXT K
```

el primero se ejecutará sin error y nos imprimirá en pantalla los números del 1 al 1Ø. Sin embargo, el segundo, tras imprimir un 1 (siempre se ejecuta una vez), sacará el mensaje, "? NEXT WITHOUT FOR ERROR IN 3Ø"; pues se ha encontrado con un NEXT K sin que hayamos empezado ningún FOR cuya variable índice se llame K. Para obviar estos errores podemos prescindir de poner el nombre de variable tras el NEXT y el Commodore se encargará de poner atención. Así, los dos programas siguientes producen el mismo resultado:

```
10 FOR I=1 TO 10
20 PRINT"HOLA"
30 NEXT I
```

```
10 FOR I=1 TO 10
20 PRINT"HOLA"
30 NEXT
```

Sin embargo, es conveniente poner el nombre de la variable tras el NEXT, pues esto mejora la legibilidad y entendimiento de nuestros programas.

Al introducir la «estructura general» habrá observado una palabra, STEP, que no habíamos mencionado antes. El valor «incremento» colocado tras el STEP indica de cuánto en cuánto irá aumentando la variable cuando pasemos por el NEXT. Considere los programas:

```
10 FOR I=1 TO 10 STEP 2
20 PRINT I
30 NEXT I
```

```
10 FOR K=1 TO 4 STEP 0.5
20 PRINT K
30 NEXT K
```

Como en el primer programa indicamos un «incremento» de 2, la línea 2Ø imprimirá en la pantalla la secuencia 1,3,5,7 y 9 (el 11 no lo imprime, pues es mayor que el 1Ø), mientras que en el segundo programa, al ser el valor de «incremento» igual a Ø5, la línea 2Ø imprimirá la secuencia 1,1.5,2,2.5,3,3.5,4. Es importante recordar con qué valor sale la variable índice de un bucle. Añada a los programas anteriores las líneas:

40 PRINT "EL VALOR DE I ES"; I
40 PRINT "EL VALOR DE K ES"; K,

observará que en el primer caso sale un 11 (pues es mayor que 1Ø), mientras que en el segundo sale un 4.5 (pues es mayor que 4).

El valor inicial no tiene que ser inferior al valor final, es decir, podemos escribir bucles decrecientes.

```
10 FOR H=10 TO 1 STEP -1
20 PRINT H
30 NEXT H
```

```
10 FOR R=-50 TO -20 STEP 10
20 PRINT R
30 NEXT R
```

En el primer programa obtendremos la secuencia 10, 9, 8,..., 2, 1, mientras que en el segundo la secuencia será —50, —40, —30, —20. Observe que el valor de H al salir del bucle es H = 0 (introduzca el comando directo PRINT H), y que el valor final de R es —10 (introduzca el comando PRINT R). Es decir, el bucle termina cuando el valor de la variable (en este caso H o R) sobrepasa el valor final, ya sea en sentido creciente o decreciente.

Y además, como indica la definición del FOR, también podemos utilizar expresiones aritméticas en valor inicial, valor final. Así, el programa siguiente le pregunta qué valor inicial, N, y qué valor final, M, desea utilizar. El incremento se calcula como (M-N)/20. Observe, para distintos valores de M y N, cómo funciona el incremento.

```
10 REM DEFINIENDO UN BUCLE
20 INPUT "VALOR INICIAL";N
30 INPUT "VALOR FINAL";M
40 FOR I=N TO M STEP(M-N)/20
50 PRINT I
60 NEXT I
```

Recuerde que «valor inicial», «valor final» e «incremento» sólo se evalúan al principio del bucle. Después, la sentencia FOR realizará la ejecución con los valores calculados.

¿Qué le parece si tratamos de modificar el programa de «Tablas de multiplicar» para utilizar el bucle FOR?

```
10 REM TABLAS DE MULTIPLICAR
11 REM
12 REM
20 REM A=ACIERTOS
30 PRINT"J"
40 INPUT "QUE NUMERO DESEA REPASAR";N
50 A=0
60 FOR C=1 TO 10
70 PRINT:PRINT:PRINT
80 PRINT N;"*";C;"=";
90 INPUT R
100 IF R=N*C THEN PRINT TAB(9);"BIEN";:
    A=A+1
110 IF R<>N*C THEN PRINT TAB(9);"MAL";
120 PRINT N;"*";C;"=";N*C
130 NEXT C
140 IF A<7 THEN PRINT "MUY MAL. HAY QUE
    VOLVER A REPASARLA":GOTO 50
150 PRINT "BIEN, DESEA REPASAR OTRA TABLA
    (S/N)"
160 GET R$: IF R$<>"S" AND R$<>"N" THEN
    160
170 IF R$="S" THEN 30
180 PRINT"J■■■■■■■■■"
190 PRINT TAB(15);"HASTA OTRA"
200 END
```

ENTRADAS Y SALIDAS EN BUCLES **FOR-NEXT**

Como se ha explicado, la salida normal de un bucle FOR-NEXT es a la línea siguiente a la sentencia NEXT, cuando se rebase el valor final de la variable. No obstante, es posible salir de un bucle FOR-NEXT antes de que se complete, normalmente con una condición.

El programa que proponemos a continuación pediría hasta 10 datos y los sumaría, a no ser que le introduzcamos un número negativo; en este caso, se sale del bucle, deja de pedir datos y presenta suma de los números introducidos antes del negativo.

```
10 FOR N=1 TO 10
20 INPUT "NUMERO?";D
30 IF D<0 THEN 100
40 S=S+D
50 NEXT N
100 PRINT"LA SUMA DE LOS NUMEROS ES";S
```

Ejecútelo y experimente con él todas las posibilidades; no obstante introducir una condición dentro de un bucle FOR-NEXT, indica que dicho bucle no es la forma óptima de resolver el problema; además, la condición de finalización del bucle puede no ser obvia para quien no ha construido el programa.

En el ejemplo anterior, si nunca metemos un número negativo el bucle acabaría por su vía normal, sólo al introducir un número negativo saldría del bucle antes de concluir éste; evidentemente, si se hace un programa de este tipo es porque no estamos seguros de la cantidad de datos que necesitamos en cada ejecución y sería más lógico diseñar el siguiente programa:

```
10 INPUT "CUANTOS NUMEROS DESEA INTRODUCIR";A
20 FOR N=1 TO A
30 INPUT "NUMERO?";D
40 S=S+D
50 NEXT N
60 PRINT"LA SUMA DE LOS NUMEROS ES";S
```

Ahora podemos fijar en cada ejecución el número de datos a sumar, y el bucle finaliza siempre por su vía normal.

Lo que nunca debemos hacer es entrar en un bucle saltándonos la sentencia FOR, porque obtendríamos un mensaje de error.

Al diseñar el programa siguiente se pretende que la suma que se realiza en la línea 40 (A=A+I) empiece desde el valor inicial I=1 si el valor que introducimos para A es positivo, y que empiece con valor de I=10 si el valor que damos a A es negativo; en este segundo caso nos meteríamos en el bucle por otro punto que la sentencia FOR, tecléelo:

```
10 INPUT "VALOR PARA A?";A
20 IF A<0 THEN I=10: GOTO 40
30 FOR I=1 TO 50
40 A=A+I
50 NEXT I
60 PRINT A
```

Al ejecutarlo comprobará cómo la ejecución transcurre con normalidad, siempre que demos a A un valor positivo, pero si introduce un número negativo, obtendrá el siguiente mensaje de error:

NEXT WITHOUT FOR ERROR IN 50

Bucles anidados

Es también muy corriente que se necesite repetir un proceso que a su vez consiste en repetir otros procesos. Esto puede conseguirse fácilmente en BASIC, basta con incluir un bucle FOR-NEXT dentro de otro bucle FOR-NEXT.

Resulta un trozo de programa con el aspecto:

```
┌─FOR variable -1=valor inicial -1 TO valor final -1 STEP incremento 1
│┌FOR variable -2=valor inicial -2 TO valor final -2 STEP incremento 2
││
│└NEXT variable -2
└─NEXT variable -1
```

A la inclusión de un bucle dentro de otro suele dársele el nombre de «anidamiento» de bucles. La utilidad de poder anidar bucles FOR-NEXT quedará patente con un ejemplo. El siguiente programa cuenta 47 huevos:

```
10 REM RECUENTO DE HUEVOS
20 FOR I=0 TO 3
30 FOR J=0 TO 11
40 PRINT I;"DOCENAS Y";J;"HUEVOS"
50 NEXT J
60 NEXT I
```

Como habrá podido observar, para cada valor de la variable índice de bucle «externo» I (0, 1, 2, ó 3), la variable índice del bucle «interno», J, recorre todos sus valores, esto es, de 0 al 11. Cuando J llega a 12 hemos contado otra docena y, por tanto, I debe aumentar en una unidad y J debe de pasar a 0.

Un punto importante a tener en cuenta, cuando se anidan bucles, es que el bucle más interno debe estar completamente contenido dentro del más externo. Teclee y ejecute el siguiente programa:

```
10 FOR I=1 TO 4
20 FOR J=2 TO 5
30 PRINT I,J
40 NEXT J
50 NEXT I
```

Si en lugar de anidarlos, se solapan, obtendremos un error; compruébelo introduciendo y ejecutando el siguiente programa.

```
10 FOR I=1 TO 10
20 FOR J=2 TO 7
30 PRINT I,J
40 NEXT I
50 NEXT J
```

Obtendrá:

```
 1  2
 2  2
 3  2
 4  2
 5  2
 6  2
 7  2
 8  2
 9  2
10  2

? NEXT WITHOUT FOR ERROR IN 50
READY
```

Una manera de garantizar que el bucle más interno se concluye antes que el más externo es omitir el nombre de la variable en la sentencia NEXT; el C-64 recordará por usted cuál es el bucle que corresponde cerrar en cada punto y ejecutará correctamente cualquier anidamiento de bucles. No obstante, recuerde que el funcionamiento de un programa será más claro si emplea la sintaxis completa de NEXT.

Otro punto a tener en cuenta, por evidente que parezca, es que la variable índice de un bucle debe tener un nombre distinto del de las variables índice de cada uno de los bucles más externos que él. A veces se olvida este aspecto y, como en el programa que ahora se le propone, utiliza nombres de variables índice que comienzan por los mismos dos caracteres:

```
10 FOR IND1=1 TO 5
20 FOR IND2= 1 TO 3
30 PRINT IND1,IND2
40 NEXT IND2
50 NEXT IND1
```

Muchas veces el rango del bucle interno es controlado por el valor de la variable índice del bucle externo. Así, el programa que aparece a continuación, que presenta los cuatro primero múltiplos de cada uno de los nueve primeros números enteros, utiliza la variable N para determinar completamente el valor inicial, el final y el incremento en el bucle interno.

```
10 FOR N=1 TO 9
20 FOR M=N TO 4*N STEP N
30 PRINT M,
40 NEXT M
50 NEXT N
```

Al ejecutar el programa obtendríamos:

1	2	3	4
2	4	6	8
3	6	9	12
4	8	12	16
5	10	15	20
6	12	18	24
7	14	21	28
8	16	24	32
9	18	27	36

Terminaremos el capítulo con nuevos ejemplos que muestran la generalidad del uso de bucles FOR-NEXT:

El ordenador va presentando ocho sumas (siempre las mismas) y pregunta el resultado; si le damos el correcto, lo indica, y si no lo es, dice que es incorrecto y presenta la respuesta correcta; en cualquier caso aparece otra suma hasta acabar con las ocho.

```
10 REM SUMAS FIJAS
20 FOR I=1 TO 2
30 FOR K=4 TO 1 STEP-1
40 PRINT I;"+";K;"=":INPUT "CUANTO
   VALE ESTA SUMA";B
50 IF B=I+K THEN PRINT"CORRECTO"
60 IF B<>I+K THEN PRINT"INCORRECTO,LA
   RESPUESTA CORRECTA ES ";I+K
70 NEXT K
80 NEXT I
```

Compruebe que podríamos sustituir las líneas 70 y 80 por una única línea 70 que fuese NEXT K,I

```
10 REM SIMULA UN RELOJ DIGITAL
20 FOR J=0 TO 59
30 FOR I=0 TO 9
40 PRINT"[illegible]";I;
   ":";I
50 FOR Z=1 TO 100:NEXTZ
70 NEXT I:NEXT J
```

Al ejecutar el programa observa «un reloj» en el centro de la pantalla.

Los caracteres de la línea 40 son ya conocidos, el primero, «[illegible]», sirve para borrar la pantalla, y los otros, [illegible], para situar la impresión en el centro de la pantalla aproximadamente. Observe también cómo en la línea 50 utilizamos un bucle FOR-NEXT para que exista un pequeño intervalo de tiempo entre cada impresión.

3 Programación avanzada

En este capítulo vamos a examinar ciertas ayudas y técnicas que nos suministra el C-64 para mejorar y hacer útiles nuestros programas. A medida que avancemos los programas van siendo, necesariamente, más largos y complicados. Es importante que los teclee y compruebe por sí mismo su funcionamiento (no se conforme con una ojeada), además es posible que alguno de ellos le llegue a ser útil en algún momento y le interese grabarlo.

FUNCIONES Y VARIABLES INCORPORADAS

En nuestra actividad cotidiana estamos acostumbrados a utilizar cierto número de *funciones* tales como «la raíz cuadrada de un número», «el seno de un ángulo», etc. Es evidente que no es difícil desarrollar un método para utilizarlas en nuestros programas, pero su uso es tan frecuente que el C-64 se ha preocupado de ofrecernos gran número de ellas incorporadas en el sistema.

Y ¿qué es una función? Observemos el ejemplo siguiente:

Y = f(x)	Y = x*x
"Y depende de x"	"Y es igual al cuadrado de x"

En la primera parte aparece la expresión general de función: «Y es función de x», que es lo mismo que decir: «Y depende de x». Esto es, el valor que toma x influirá en el valor que tome Y. x recibe el nombre de *argumento* e Y es el *resultado* de aplicar sobre x la función f. Quizá todo esto es un poco confuso. Vamos a explicarlo mediante la segunda parte del ejemplo. Aquí tenemos unos casos particulares:

Función	*Argumento* *x*	*Resultado* *Y*
Y = x*x	2	4
	3	9
	4	16
	.	.
	.	.
	.	.

En el Commodore vienen incorporadas gran número de funciones. Sin embargo, existe una limitación en la capacidad de manejo de funciones por los

ordenadores: *las funciones han de dar un resultado único.* Es decir, para un valor del argumento x sólo pueden dar un valor del resultado Y. Esto, como veremos, afecta a funciones como SQR (x), (raíz cuadrada de X), pues la raíz cuadrada de un número tiene dos resultados: por ejemplo, la raíz cuadrada de 4 puede ser o bien −2 o bien +2, sin embargo la función SQR sólo presenta uno, el positivo.

Además de las *funciones numéricas* (aquellas en que el argumento y el resultado son datos numéricos), el C-64 incorpora diversas funciones para trabajar con datos de tipo alfanumérico y, naturalmente, reciben el nombre de *funciones alfanuméricas*.

Vamos a realizar una revisión de todas estas posibilidades que nos darán mucha potencia y agilidad en la construcción de nuestros programas.

Funciones alfanuméricas

En el capítulo 2 vimos los datos y las variables alfanuméricas, pero hasta ahora lo único que sabemos hacer con ellas es introducirlas, imprimirlas y poco más aparte de la concatenación de cadenas. Vamos a ver qué más nos ofrece el Commodore para enfrentarnos con este tipo de datos. Esta sección es muy importante, pues muchos de los datos con los que tendremos que trabajar (ficheros) serán de tipo alfanumérico.

La función LEN

La estructura de la función LEN es:

LEN (A$)

A$: representa un argumento alfanumérico, puede ser una constante, una variable o una expresión alfanumérica.

Esta función nos permite conocer el número de caracteres del argumento. Así, con el programa:

```
10 A$="HOLA":B$="?QUE TAL ESTAS?"
20 PRINT LEN(A$),LEN(B$),LEN(A$+B$)
```

Aparecerán en pantalla los números 4, 15 y 19. El primer y tercer caso son evidentes. En el segundo hemos de tener en cuenta que también los caracteres en blanco forman parte de la longitud de la cadena. La frase «¿Qué tal estás?» contiene dos símbolos de interrogación, dos espacios en blanco y 11 letras. En total 15 caracteres.

Esto nos sirve para poder manejar cualquier cadena que nos encontremos, sin tener que conocer previamente su longitud. Pero, antes de construir ejemplos más útiles, vamos a estudiar las funciones de fragmentación de cadenas.

Fragmentación de cadenas

Recibe este nombre la operación de extraer un conjunto de caracteres de una cadena, y para realizarla disponemos de tres funciones, que son: LEFT$ (left = izquierda), MID$ (middle = medio) y RIGHT$ (right = derecha).

La estructura de la función LEFT$ es:

LEFT$ (A$, n)

A$: Representa un argumento alfanumérico.
n: Indica el número de caracteres que se van a extraer.

La lectura completa de LEFT$ (A$, n) podría ser «extrae los n primeros caracteres comenzando por la izquierda». Por ello, la ejecución de la línea:

10 PRINT LEFT$ ("MARIA", 3)

Imprimirá los caracteres MAR en la pantalla. El programa:

```
10 PRINT "□"
20 INPUT "TECLEA UNA PALABRA";N$
30 FOR I=1 TO LEN(N$)
40 PRINT LEFT$(N$,I)
50 NEXT I
```

va imprimiendo, sucesivamente el primer carácter de N$, los dos primeros, los tres primeros, etc. Termina por escribir completamente N$.

El comportamiento de la función:

RIGHT$ (A$, n)

A$: Representa un argumento alfanumérico.
n: Número de caracteres que se van a extraer.

Podríamos decir que es el inverso, pues, comenzando por la derecha, extrae los n primeros caracteres. Compruébelo, introduciendo el programa:

```
10 PRINT "□"
20 INPUT "TECLEA UNA PALABRA";N$
30 FOR I=1 TO LEN(N$)
40 PRINT RIGHT$(N$,I)
50 NEXT I
```

La función MID$ es la más complicada. Tiene tres argumentos.

MID$ (A$, p, n)

A$: Representa un argumento alfanumérico.
p: Representa la posición del primer carácter que se va a extraer.
n: Es el número de caracteres, hacia la derecha, que se van a extraer.

Para el ordenador, los caracteres que forman una cadena están numerados de izquierda a derecha, empezando por el 1. De este modo, en:

«ME LLAMO COMMODORE»

la letra A ocupa la posición 6, la letra C la 10 y la letra D la 15. Si introduce el siguiente comando directo:

PRINT MID$ ("ME LLAMO COMMODORE", 10, 3)

en la pantalla aparecerá COM, pues la C ocupa la décima posición, y hemos indicado que se extraigan tres caracteres. Recuerde que también los espacios en blanco son caracteres.

El programa:

```
10 PRINT "♥"
20 INPUT "TECLEA UNA PALABRA";N$
30 FOR I=LEN(N$) TO 1 STEP-1
40 PRINT MID$(N$,I,1);
50 NEXT I
60 GO TO 60
```

se encarga de escribir al revés cualquier palabra que usted teclee. Observe cómo se ha utilizado un bucle FOR decreciente para que el valor de I comience por la última posición de carácter de N$ y vaya hasta la primera. Otro punto a observar es la utilización en la línea 60 del bucle infinito.

60 GO TO 60

Para evitar la aparición del mensaje READY con el que el Commodore nos obsequia, cada vez que termina la ejecución. Para salir de este bucle pulse la tecla RUN/STOP.

Antes de abandonar estas funciones trate de deducir lo que imprimiría este programa:

```
10 A$="CUCHARA"
20 PRINT LEFT$(A$,3)+RIGHT$(A$,3)+
   MID$(A$,3,3)
```

¿Y éste?:

```
10 A$="MUCHACHOS"
20 B$=RIGHT$(A$,1)+MID$(A$,8,1)+
   LEFT$(A$,1)+MID$(A$,8,2)
30 C$=LEFT$(A$,2)+RIGHT$(A$,4)
40 PRINT B$+" "+C$+" "+A$
```

En las sentencias PRINT, de ambos ejemplos, podríamos haber sustituido el símbolo "+" (concatenación de cadena) por el símbolo ";".

Si omitimos el tercer parámetro del MID$, n, obtendremos todos los caracteres que haya en la cadena, desde el indicado por p hasta el final.
Si en MID$ omitimos los argumentos n y p obtendremos un mensaje de error. Si en LEFT$ y en RIGHT$, omitimos n, el número de caracteres obtendremos un mensaje de error.
Si en LEFT$, RIGHT$ y MID$, ponemos un valor de n, superior al número de caracteres que se pueden tomar, obtendremos todos estos caracteres.

LAS FUNCIONES VAL Y STR$

Son, en cierto sentido, simétricas. VAL convierte una cadena, que contenga dígitos, en un número, mientras que STR$ convierte un número en una cadena. Veamos algunos ejemplos:

```
10 REM LA FUNCION VAL
20 A$="12":B$="34"
30 PRINT A$+B$
40 A=VAL(A$):B=VAL(B$)
50 PRINT A+B
```

Éste es un buen momento para repasar la diferencia entre la cadena «12» y el número 12. Cuando tenemos una serie de dígitos, por ejemplo 12, dentro de una cadena, el ordenador no lo interpreta como el número 12, sino como los caracteres 1 y 2. Por ello, cuando el control llega a la línea 30 en la pantalla aparece la cadena 1234, la unión de ambas cadenas, y no el resultado de sumar los números 12 y 34. Quizá esto hubiera sido más claro si hubiéramos concatenado las cadenas A$=«HOLA» y B$=«PEPE». Cuando el control llega a la línea 40, la función VAL saca los dígitos contenidos en las cadenas A$ y B$ y los introduce, como números, en las variables A y B. Finalmente, la línea 50 imprime el número 46 como resultado de sumar (esta vez sí) los números 12 y 34. El formato general de la función es:

VAL (A$)

A$: Argumento alfanumérico.
Si A$ contiene sólo dígitos, por ejemplo, A$="346", B=VAL (A$) es el número 346.
Si A$ contiene espacios en blanco delante de los dígitos, VAL los ignora y extrae los dígitos.
Si delante de los dígitos hay un signo, se extrae también el signo.
Si A$ contiene letras y dígitos, pueden darse dos casos:
a) Empieza por dígitos: entonces VAL saca todos los dígitos que haya hasta la primera letra.
b) Empieza por letra: en este caso VAL nos devuelve un cero.

¿Y cómo actúa STR$?, pues justo al revés.

```
10 REM LA FUNCION STR$
20 A=12:B=34
30 PRINT A+B
40 A$=STR$(A):B$=STR$(B)
50 PRINT A$+B$
```

En la línea 20 introducimos los números 12 y 34 en las variables A y B, respectivamente. Después en la 40 utilizamos STR$ sobre A y B para convertir los números que representan en las cadenas 1, 2 y 3, 4, que introducimos en las variables A$ y B$. De este modo, la línea 50 imprimirá en la pantalla la secuencia 1234. La estructura de la función STR$ es:

STR$ (A)

A: Argumento numérico.
Tenga cuidado, pues STR$ reserva un espacio para el signo de A. Así, si hacemos A=3 y A$=STR$ (A), el comando PRINT LEN (A$) sacará un 2 en la pantalla, pues la primera posición de A$ se reserva para el posible signo de A.

Las funciones CHR$ y ASC

El ordenador almacena la información, en su memoria, mediante unos códigos numéricos. Estos códigos, en sistema decimal, van del 0 al 255, esto es, 256 códigos distintos. Por ejemplo, a la letra A le corresponde el código 65; a la B, el 66, y así, hasta la Z, a la que le corresponde el 90. Al dígito 0 le corresponde el código 48, al 1 el 49 y así hasta el 9, al que le corresponde el código 57 (vea el apéndice D). Compruébelo ejecutando el programa:

```
10 REM CODIGOS DE LAS LETRAS
20 FOR I=65 TO 90
30 PRINT CHR$(I),
35 FOR K=1 TO 500:NEXT K
40 NEXT I
```

Que imprimirá en la pantalla las letras de la A a la Z. En la línea 30 vemos la utilización de la función CHR$. CHR$ tiene un argumento numérico, I, y da como resultado un carácter: el correspondiente al código indicado, I. Así, utilizando CHR$ podemos saber a qué carácter corresponde cierto código. Es fácil resumir su estructura general.

CHR$ (A)

A es un argumento numérico cuyo valor puede variar de 0 a 255. Un valor fuera de este rango provocará un mensaje de error.

Mediante CHR$ podemos manejar caracteres que no pueden aparecer en cadenas. Por ejemplo, si queremos imprimir un texto con comillas podemos utilizar:

```
10 INPUT"TECLEA UNA PALABRA";P$
20 PRINT "J"
30 PRINT CHR$(34);P$;CHR$(34)
```

ASC es la función inversa. Esto es, el argumento es un carácter y responde con el código correspondiente a dicho carácter. Considere el ejemplo siguiente:

```
10 REM ***********************
20 REM * DETECTOR DE TRAMPAS *
30 REM ***********************
40 PRINT "J"
50 PRINT:PRINT:PRINT
60 PRINT TAB(7);"PULSE UN NUMERO"+
   " DEL 1 AL 6"
70 GETA$:IF A$="" OR A$=CHR$(13)
   THEN70
80 IF ASC(A$)<49 OR ASC(A$)>54 THEN
   PRINT:PRINT:PRINT TAB(16);"TRAMPOSO"
90 PRINT:PRINT:PRINT
100 PRINT TAB(9);"HA PULSADO LA TECLA "
    ;A$
```

En la línea 70 revisamos el teclado para ver si se ha pulsado alguna tecla. Observe que si hubiéramos empleado GET A o GET A$ y el usuario pulsase una tecla de letra, obtendría un mensaje de error. Utilizamos GET A$, pues así podemos teclear letras o números y después, usando ASC, averiguaremos cuál es el carácter que ha introducido. En la segunda parte de la línea 70 la comparación A$= "" obliga a esperar a que se pulse alguna tecla, y la comparación A$=CHR$ (13) se encarga de no aceptar que la tecla pulsada sea la tecla RETURN. (El código de la tecla RETURN es el 13.)

Después, en la línea 80 comprobamos si el código de la tecla pulsada está

comprendido entre el 49 y el 54, que son los correspondientes a los dígitos 1 y 6, respectivamente. Si no es así, el Commodore le acusará de marrullero.

El esquema de este programa es ampliamente utilizado para evitar que nadie pueda hacerse el listillo y estropear la ejecución del programa por haber pulsado una tecla distinta de las que el ordenador esperaba recibir.

La estructura general de la función ASC es:

ASC (A$)

A$: Argumento alfanumérico. Si el argumento contiene más de un carácter, ASC devolverá el código correspondiente al primer carácter de la cadena. Por ejemplo:

PRINT ASC ("COMMODORE")

Imprimirá en la pantalla el número 67, que es el código correspondiente a la "C".

El reloj del tiempo real

Una característica del C-64 es que mantiene en funcionamiento un reloj. Un reloj en horas, minutos y segundos que podemos utilizar para saber cuánto tiempo hace que estamos trabajando o cuánto tiempo tarda el ordenador en realizar ciertas cosas.

A este reloj podemos acceder mediante dos variables TI$ (TIME$) y TI (TIME). Cuando enchufamos el ordenador estas dos variables comienzan en 0 y van incrementándose como lo hace un reloj. Pero podemos manejar el contenido de estas variables sin necesidad de apagar y encender el Commodore.

El contenido de la variable TI$ es de la forma:

TI$= "hhmmss"

Donde: hh representa las horas, de 0 a 23.
mm representa los minutos, de 0 a 59.
ss representa los segundos, de 0 a 59.

Cuando encendemos el ordenador TI$ comienza con el contenido "000000" y va variando de segundo en segundo. Si queremos darle a TI$ cierto valor, por ejemplo, las 9 de la mañana, 15 minutos y 40 segundos, hemos de introducir el comando:

TI$= "091540"

Trate de utilizar TI$ para construir un reloj en el centro de la pantalla y compare después su solución con el programa siguiente:

```
10 REM ****************
20 REM * RELOJ DIGITAL *
30 REM ****************
32 PRINT "♥"
34 REM *******************
36 REM * INTRODUCE HORAS *
38 REM *******************
40 INPUT"TECLEE LA HORA CON 2 CIFRAS";H$
50 IF LEN(H$)<>2 THEN 40
60 IF LEFT$(H$,1)="2"AND
   ASC(RIGHT$(H$,1))>51 THEN 40
70 IF ASC(LEFT$(H$,1))<48 OR
   ASC(LEFT$(H$,1))>50 THEN 40
80 IF ASC(RIGHT$(H$,1))<48 OR
   ASC(RIGHT$(H$,1))>57 THEN 40
82 PRINT "♥"
84 REM *********************
86 REM * INTRODUCE MINUTOS *
88 REM *********************
90 INPUT"TECLEE MINUTOS CON 2 CIFRAS";M$
100 IF LEN(M$)<>2 THEN 90
110 IF ASC(LEFT$(M$,1))<48 OR
    ASC(LEFT$(M$,1))>53 THEN 90
120 IF ASC(RIGHT$(M$,1))<48 OR
    ASC(RIGHT$(M$,1))>57 THEN 90
122 PRINT "♥"
124 REM **********************
126 REM * INTRODUCE SEGUNDOS *
128 REM **********************
130 INPUT"TECLEE SEGUNDOS CON 2 CIFRAS";
    S$
140 IF LEN(S$)<>2 THEN 130
150 IF ASC(LEFT$(S$,1))<48 OR
    ASC(LEFT$(S$,1))>53 THEN 130
160 IF ASC(RIGHT$(S$,1))<48 OR
    ASC(RIGHT$(S$,1))>57 THEN 130
162 REM ***********************
164 REM * CRECION E IMPRESION *
166 REM *      DEL RELOJ      *
168 REM ***********************
170 TI$=H$+M$+S$
180 R=VAL(TI$)
190 PRINT "♥QQQQQQQQQQQ"
200 PRINT TAB(15);LEFT$(TI$,2);":";
    MID$(TI$,3,2);":";RIGHT$(TI$,2)
210 IF R-VAL(TI$)=0 THEN 210
220 GO TO 180
```

Complicado, ¿no? En realidad es bastante sencillo, la complicación surge de querer hacer un programa que impida que el usuario introduzca datos incorrectos. Las líneas 5Ø-8Ø se dedican a comprobar que usted teclea una hora comprendida entre ØØ y 23; y si no, no le dejará pasar de aquí. Si el primer dígito de la hora es el Ø ó el 1, el segundo dígito puede variar de Ø a 9; pero si el primer dígito es un 2, entonces el segundo sólo puede tomar los valores Ø, 1, 2, ó 3, y por esto surge la línea 6Ø. Las líneas 1ØØ-12Ø y 14Ø-16Ø se dedican a comprobar que tanto los minutos como los segundos estén comprendidos entre ØØ y 59. Estas líneas de comprobación evitan que se pueda introducir cualquier otro rango de números o que se introduzcan letras, en lugar de números, para ¡a ver qué pasa! Esto es, hemos dedicado más de la mitad del programa a verificar que el usuario inicializa el reloj en forma adecuada.

El reloj se construye y se pone en funcionamiento con las líneas 17Ø-22Ø. En la línea 17Ø se inicializa el reloj interno de la máquina con la hora que hemos introducido, y en la línea 2ØØ se imprimen las horas, minutos y segundos, separados por símbolos de dos puntos para mejorar la legibilidad. ¿Y para qué sirven las líneas 18Ø, 21Ø y 22Ø? En la línea 18Ø introducimos el valor inicial de TI$ en R, y con la línea 21Ø obligamos a que no se vuelva a imprimir el reloj hasta que haya pasado un segundo, es decir, hasta que el valor actual de TI$ sea distinto del valor anterior de TI$ (que está metido en R). Después, en la línea 22Ø se introduce en R el nuevo valor de TI$ para poder seguir comparando.

Desde luego, hemos hecho un uso extensivo de las funciones de cadena. Trate de reducir esta complicación utilizando variables numéricas H, M y S en lugar de H$, M$, y S$.

Además de esta variable alfanumérica TI$ disponemos de otra variable real, TI, para poder utilizar el reloj del ordenador. TI puede tomar valores entre Ø y 5184ØØØ y aumenta en una unidad cada 1/6Ø segundos. Es decir, cada vez que TI ha aumentado en 6Ø ha pasado un segundo. Así, 24 horas, que es lo máximo que podemos representar, tanto con TI como con TI$, es igual a:

```
24*6Ø*6Ø*6Ø = 5184ØØØ
 |   |  |
 |   |  └─unidad de tiempo del TI
 |   └────segundos
 └────────minutos
```

Cuando encendemos el ordenador, tanto TI$ como TI comienzan desde Ø. Cuando asignamos un valor TI$, también se lo asignamos a TI, así si ejecutamos:

```
1Ø TI$="143226"
2Ø PRINT TI$,TI
```

En la línea 1Ø ponemos TI$ en 14 horas, 32 minutos y 26 segundos, y por ello el valor de TI es 14*6Ø*6Ø*6Ø+32*6Ø*6Ø+26*6Ø = 314Ø76Ø.

Una aplicación interesante de estas dos variables es la de utilizarlas para provocar pausas controladas en los programas. Así, las líneas siguientes podemos utilizarlas en muchos de nuestros programas para provocar una pausa de 2 segundos.

```
1Ø REM PROVOCANDO UNA PAUSA
15 PRINT "□"
2Ø PRINT TAB(1Ø);"PAUSA DE DOS SEGUNDOS"
3Ø TI$="ØØØØØØ"
4Ø IF VAL(TI$)<2 THEN 4Ø
5Ø PRINT "□"
6Ø PRINT TAB(15);"SE ACABO"
```

```
1Ø REM PROVOCANDO UNA PAUSA
15 PRINT "□"
2Ø PRINT TAB(1Ø);"PAUSA DE DOS SEGUNDOS"
3Ø TI$="ØØØØØØ"
4Ø IF TI<12Ø THEN 4Ø
5Ø PRINT "□"
6Ø PRINT TAB(15);"SE ACABO"
```

Si se desea pausas más prolongadas, aumente el valor de comparación de la línea 4Ø. Observe, en el segundo programa, que para darle un valor inicial a TI lo hemos de hacer a través de TI$ (línea 3Ø). Si intenta inicializar directamente TI, obtendrá un mensaje de error.

Otra aplicación importante de estas dos variables es la de medir el tiempo que tarda en ejecutarse un programa. Miramos el valor de TI antes de empezar a ejecutar el programa, y cuando éste termina volvemos a mirar el valor de TI; la diferencia entre estos valores nos indica la duración del programa. Vamos a practicarlo midiendo la duración de las pausas controladas por el bucle FOR.

```
1Ø REM *************************
2Ø REM * DURACION DE UNA PAUSA *
3Ø REM *         FOR           *
4Ø REM *************************
5Ø PRINT "□"
6Ø INPUT"TECLEE EL VALOR FINAL DEL FOR";F
7Ø PRINT "□"
8Ø PRINT:PRINT:PRINT TAB(17);"PAUSA"
9Ø R=TI
1ØØ FOR I=1 TO F:NEXT I
11Ø X=TI-R
12Ø PRINT "□"
13Ø PRINT "ESTA PAUSA HA DURADO";X/6Ø;
    "SEGUNDOS"
14Ø PRINT:PRINT:PRINT
15Ø PRINT"PROBAMOS OTRA VEZ (S/N)?"
16Ø GET A$:IF A$="" OR A$=CHR$(13)
    THEN 16Ø
17Ø IF A$="S" THEN 5Ø
18Ø IF A$<>"N" THEN 15Ø
19Ø PRINT:PRINT:PRINT TAB(15);
    "HASTA OTRA"
2ØØ GO TO 2ØØ
```

Observe, en la línea 100, que este bucle no hace más que entretener el control de ejecución, mientras que I pasa de 1 a F. Así, cuanto mayor sea el valor de F más larga será la pausa que introduzcamos. La construcción de este tipo de pausas es mucho más sencilla que si utilizamos TI o TI$, y lo único que tenemos que hacer, antes de utilizarla, es tener una idea aproximada de cómo varía su duración, según los valores que le asignemos a F. En la figura 3.1 se exponen algunos de los resultados, obtenidos en diversas ejecuciones del programa.

Valor de F	*Duración aproximada en segundos*
355	0.5
715	1.0
1000	1.38333333
1460	2.0
2000	2.76666667
3000	4.18333334
4000	5.63333333

Fig. 3.1. Aproximación de la duración de ciertas pausas FOR.

Sin embargo, estos valores oscilan, y por ello es mejor que ensaye usted mismo los valores que más le interesen y construya su propia tabla de duraciones.

En la línea 90 introducimos en R el valor de TI antes de la pausa FOR. Para obtener la duración de la pausa basta con restar al valor de TI, después de la pausa, el que tenía antes (que tenemos guardado en R), y esto se realiza en la línea 110. La diferencia se introduce en X, y como TI aumenta 60 unidades por segundo, para calcular el número de segundos que ha durado la pausa hay que dividir X entre 60.

Las líneas 150—190 le dan la oportunidad de realizar otro ensayo y se ocupan de impedir que su respuesta pueda ser otra que la de pulsar la tecla S (para sí) o la tecla N (para no). Al final del programa volvemos a utilizar la estructura de bucle infinito:

200 GO TO 200

que evita que aparezca en pantalla el mensaje READY tras la finalización del programa. Para salir de este bucle pulse la tecla RUN/STOP.

Las funciones numéricas

No se asuste, en realidad no son tan terribles y nosotros nos limitaremos a explicarle cómo utilizarlas en su ordenador. La mejor o peor comprensión de ellas dependerá, fundamentalmente, del grado en que usted necesite emplear estas funciones en sus programas. Si usted está familiarizado con estas funciones, seguramente le bastará con revisar la descripción general que se realiza en el apéndice B, y si es un «matematífugo», tenga la seguridad de que podrá construir muy buenos programas sin necesidad de aprender matemáticas. Para el estudio de estas funciones las agruparemos en dos bloques: las funciones aritméticas y las funciones trigonométricas.

Funciones aritméticas

En este apartado revisaremos las funciones INT, ABS, SGN, SQR, LOG, EXP, y el número π.

La función INT (*integer* = entero) sirve para extraer la parte entera de un número. Esto es, si hacemos Y=INT (X), para X=0.6, Y valdrá 0, para X=3.83, Y valdrá 3, etc.

Compruébelo ejecutando el programa:

```
10 REM *****************
20 REM * LA FUNCION INT *
30 REM *****************
40 PRINT "□"
50 INPUT"TECLEE UN NUMERO";X
60 PRINT "□":PRINT:PRINT:PRINT
70 PRINT "LA PARTE ENTERA DE ";X;" ES ";
   INT(X)
80 PRINT:PRINT:PRINT:GO TO 50
```

Así, verá que la parte entera de 4.38 es 4, la de 1,99 es 1, ¿pero qué pasa si introducimos números negativos? La definición de parte entera de un número es: *el primer número entero tal que es menor o igual que el número dado*, y por ello si X= —4.27 veremos aparecer en la pantalla el número —5, y si introducimos —0.1 obtendremos —1. ¿Por qué? Acabamos de decirlo, la función INT devuelve el primer número entero cuyo valor sea menor o igual que el argumento y —1 < —0.1. Es importante que se fije en esto, pues es fácil equivocarse e interpretar que INT simplemente suprime la parte decimal del número, y es cierto que ocurre así con los positivos, pero no con los negativos. (Para salir del programa anterior, recuerde que en un INPUT hay que pulsar las teclas RUN/STOP y RESTORE.)

Esta función es muy útil para evitar respuestas absurdas en nuestros programas (no podemos comprar 1.32 coches). Su estructura general es:

INT (X)

> X: Argumento real puede ser una constante, una variable o una expresión aritmética. Es importante observar que INT toma la parte entera de un número real, pero no lo convierte de número real a número entero (X%).

La función ABS (viene de ABSOLUTE = ABSOLUTO) se utiliza para obtener el valor absoluto de un número. Valor absoluto de un número es el mismo número, pero prescindiendo del signo. Así ABS (+5)=5 y ABS (—5)=5. Creemos una máquina inteligente:

```
10 REM ****************************
20 REM * LA ADIVINADORA DE SIGNOS *
30 REM ****************************
40 PRINT"□"
50 PRINT"TECLEE UN NUMERO POSITIVO O "+
   "NEGATIVO":INPUT A
```

```
60 IF A=ABS(A) THEN PRINT A,"ES POSITIVO"
70 IF A<>ABS(A) THEN PRINT A,"ES NEGATIVO"
80 PRINT:PRINT:PRINT"ACERTE ?EH?"
90 PRINT:PRINT:PRINT"?OTRO NUMERO?"
100 GET A$
110 IF A$="S" THEN 40
120 IF A$<>"N" THEN 100
130 PRINT TAB(15);"HASTA OTRA"
```

La estructura general de ABS es:

ABS(X)

X: Argumento real; puede ser una constante, una variable o una expresión aritmética.

SGN viene de la palabra SIGN (signo) y es como el complementario de ABS; en vez de extraer el número extrae el signo. Esto es, sirve para informarnos si el signo de un número es el + ó el —. Si el número es positivo, SGN devuelve un 1; si es negativo, un —1, y si el número es el cero, devuelve un 0. Así, SGN (—5)=—1, SGN (7)=1 y SGN (0)=0. En el programa anterior podríamos alterar las líneas 60 y 70 para utilizar SGN en lugar de ABS, quedando en la forma:

```
10 REM *******************************
20 REM * OTRA ADIVINADORA DE SIGNOS *
30 REM *******************************
40 PRINT"□"
50 PRINT"TECLEE UN NUMERO POSITIVO O "+
   "NEGATIVO":INPUT A
60 IF SGN(A)=0 THEN PRINT A,
   "ES EL CERO"
65 IF SGN(A)=1 THEN PRINT A,
   "ES POSITIVO"
70 IF SGN(A)=-1 THEN PRINT A,
   "ES NEGATIVO"
80 PRINT:PRINT:PRINT"ACERTE ?EH?"
90 PRINT:PRINT:PRINT"?OTRO NUMERO?"
100 GET A$
110 IF A$="S" THEN 40
120 IF A$<>"N" THEN 100
130 PRINT TAB(15);"HASTA OTRA"
```

El formato de SGN es:

SGN (X)

X: Argumento real, puede ser una constante, una variable o una expresión aritmética.

Y no podría faltar la función SQR. ¡Esto parece una calculadora! SQR proviene del inglés SQUARE ROOT (raíz cuadrada) y su empleo es de sobra conocido. Sin embargo, no está de más que recordemos dos puntos: uno, debido a las particularidades de esta función, y otro, debido a las particularidades de los ordenadores. La primera es, acuérdese, que no podemos hallar directamente la raíz cuadrada de un número negativo (el ordenador no está preparado, todavía, para trabajar con campos imaginarios), y si trata de hacerlo, obtendrá un mensaje de error: ILLEGAL QUANTITY ERROR. Es fácil observar que no existe un número real que, multiplicado por sí mismo, nos dé un número negativo (—2*—2=4).

El otro punto lo comentábamos al principio de esta sección. El ordenador no puede tomar decisiones por sí mismo, y si le decimos que la raíz cuadrada de 4 puede ser bien +2 o bien —2, ¿qué hace?, ¿con cuál se queda? En principio, el computador no puede enfrentar cuestiones de este tipo y sólo es capaz de manejar funciones cuyo resultado sea único. Hasta ahora todas las funciones que hemos visto eran de este tipo. Pues bien, en este caso la forma de resolver el problema es muy simple: siempre toma como resultado la raíz positiva. Así, SQR (4)=2, SQR (9)=3, etc.

```
10 REM ******************
20 REM * LA FUNCION SGN *
30 REM ******************
40 PRINT"□"
50 INPUT"TECLEE UN NUMERO";N
55 PRINT:PRINT:PRINT
60 IF SGN(N)=-1 THEN PRINT"NO PUEDO "+
   "CALCULAR NUMEROS NEGATIVOS":GOTO 80
70 PRINT"LA RAIZ CUADRADA DE ";N;
   " ES ";SQR(N);" O ";-SQR(N)
80 PRINT:PRINT:PRINT"?OTRO NUMERO?(S/N)"
90 GET A$
100 IF A$="S" THEN 40
110 IF A$<>"N" THEN 90
120 PRINT"□";TAB(15);"HASTA OTRA"
```

Observe cómo en la línea 60 utilizamos la función SGN para evitar la introducción de números negativos. Resumiendo, la estructura a la función SQR es:

SQR (X)

X: Argumento real positivo, puede ser una constante, una variable o una expresión aritmética. ¡Cuidado con los números negativos! Originarán un mensaje de error.

Y ahora a por LOG y EXP. LOG es la abreviatura de LOGARITHM (logaritmo) ¿Y qué? Vamos a tratar de explicarlo en pocas palabras. En general, estamos acostumbrados a hablar de logaritmos decimales, y decimos que Y es igual al logaritmo decimal (en base 10) de X si 10 elevado a Y es igual a X, esto es:

$$Y = \log_{10}(X), \text{ si } 10^Y = X$$

Así, $\log_{10} 100 = 2$, pues $10^2 = 100$. En resumen, Y es la potencia a la que hay que elevar el 10 para obtener X. Claro, ¿no? Pues, para simplificar un poco más las cosas, esta máquina no trabaja con logaritmos decimales, sino con logaritmos neperianos. ¿Y qué es un logaritmo neperiano? Pues aquellos en los que la base es el número e. ¿Y qué es el número e? Vale, tranquilidad. El número e es bastante extraño. Su valor es próximo a 2.718281. Si ya lo conoce, no tendrá problemas con él, y, en caso contrario, no se preocupe, tampoco será algo frecuente en sus programas. Así que, siguiendo con los logaritmos, el logaritmo neperiano (también llamado logaritmo natural) de X es aquel valor Y tal que e elevado a Y dé como resultado X.

$$\log_e(X) = Y, \text{ si } e^Y = X$$

Si en sus cálculos necesita hallar el logaritmo decimal, utilice la expresión $LOG_{10}(X) = LOG(X)/LOG(10)$, y si quiere hallar el logaritmo en otra base cualquiera, $LOG_A(X) = LOG(X)/LOG(A)$.

La función EXP, llamada función exponencial, sirve para calcular potencias del número e: $EXP(Y) = e^Y = X$. Observe que esta función es como la inversa de LOG y así:

$$LOG(EXP\ (Y)) = LOG(e^Y) = \log_e e^Y = Y$$

Es difícil explicar en general la importancia de estas dos funciones, LOG y EXP, que son casi omnipresentes en todas las áreas técnicas, pero si acaso no las conoce, no se desanime, no son necesarias para ser un buen programador. Si se atreve, varíe el programa anterior «LA FUNCION SQR», para calcular LOG y EXP de los números que el usuario teclee. El formato general de estas dos funciones es:

LOG (X)

EXP (X)

X: Argumento real, puede ser una constante, una variable o una expresión aritmética.
LOG (X): No admite argumentos negativos ni nulos. Devolverá un mensaje de ILLEGAL QUANTITY ERROR.
EXP (X): X puede oscilar entre ± 88.029691.

Para cerrar este apartado vamos a ver algo que no es una función, el número π (pi). ¿Otro número?, pues sí, y también rarillo. π = 3.14159265, y a pesar de ser tan extraño es bastante útil. A todos nos obligaron a estudiar que el área de un círculo se calcula mediante la expresión $\pi * r^2$, donde r es el radio, y que la longitud de la circunferencia se obtiene del producto $2 * \pi * r$. Escribamos un programa calculador de áreas y longitudes:

```
10 REM **************************
20 REM * EL REY DE LOS CIRCULOS *
30 REM **************************
40 PRINT"□"
50 INPUT"TECLEE EL RADIO";R
60 IF R<0 THEN 40
70 PRINT:PRINT:PRINT
80 PRINT"EL AREA DEL CIRCULO DE RADIO"
85 PRINT R;"ES";π*R↑2
90 PRINT:PRINT:PRINT
100 PRINT"LA LONGITUD DE LA "+
    "CIRCUNFERENCIA"
105 PRINT"DE RADIO";R;"ES";2*π*R
110 PRINT:PRINT:PRINT"OTRO NUMERO?(S/N)"
120 GET A$
130 IF A$="S" THEN 40
140 IF A$<>"N" THEN 120
150 PRINT"□":PRINT TAB(15);"HASTA OTRA"
```

Y ahora a por las terribles funciones trigonométricas.

FUNCIONES TRIGONOMÉTRICAS

Todos hemos sufrido la trigonometría, y el ordenador nos va a hacer sufrir un poco más, pues para utilizar las funciones disponibles tenemos que introducir los ángulos en radianes y no en grados, que es a lo que estábamos acostumbrados. Las funciones disponibles son:

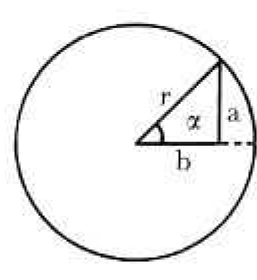
SIN (X)

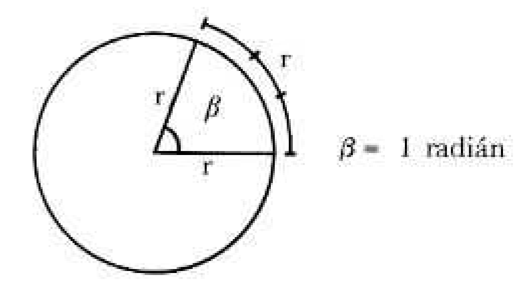
COS (X)

TAN (X)

para expresar, respectivamente, el seno, coseno y tangente del ángulo X. Pero, como acabamos de indicar, el argumento no lo podemos expresar en grados, sino en radianes. ¿Y qué es un radián? Un radian es el ángulo tal que el sector de circunferencia que abarca tiene una longitud igual a r (ver Fig. 3.2).

SIN—seno; seno $\alpha = a/r$
COS—coseno; coseno $\alpha = b/r$
TAN—tangente; tangente $\alpha = (\text{sen}\,\alpha/\cos\alpha) = a/b$





$\beta =$ 1 radián

Fig. 3.2. Brochazos de trigonometría.

La mejor forma de no equivocarse al utilizarlo es recordando que un ángulo de 9Ø° equivale a un ángulo de $\pi/2$ radianes (¡el famoso número π!), o que uno de 180° es igual a uno de π radianes. Por ello, si queremos cálcular en radianes un ángulo de α grados, utilizaremos la expresión:

$$\text{Radianes} = \alpha * \pi / 18\emptyset$$

Otra función trigonométrica disponible es:

TAN (X)

que es la abreviatura de «arco cuya tangente es X». Esto es, si introducimos un número real X nos devuelve el ángulo, en radianes, del arco cuya tangente vale X. Si usted es hábil en trigonometría podrá construir, a partir de estas funciones, toda clase de secantes, cosecantes, etc., e incluso cosenos hiperbólicos etc. Excitante, ¿no?

Y si no es hábil en trigonometría y además ha sido capaz de leer todo esto, enhorabuena y no se preocupe, no todo el libro es así, y para demostrarlo vamos a entretenernos con la relajante y misteriosa función RND. Trate de dominarla, pues le permitirá construir toda clase de juegos para disfrutar con sus amigos.

La generación de números aleatorios

La abreviatura RND viene de la palabra inglesa RANDOM (aleatorio). Con esta función podemos generar al azar (aleatoriamente) números en el intervalo (Ø, 1); esto es, cualquier número entre el Ø y el 1, pero nunca el Ø ni el 1. Por ejemplo, podemos conseguir números como el Ø827743801, Ø187534367, Ø549758526, etc.

Pero, ¿qué es eso de generar números al azar? Consideremos el lanzamiento de un dado. Sabemos que el resultado de lanzar un dado será un número entre el 1 y el 6, pero lo que no podemos afirmar es qué va a salir la próxima vez que lo lancemos. Lo único que podemos afirmar es que será un número del 1 al 6. Pues bien, esto es lo que recibe el nombre de generación de números al azar. Cuando utilizamos RND vamos obteniendo números entre el Ø y el 1, pero no sabemos qué número aparecerá la próxima vez.

Vamos a probar con un ejemplo. Introduzca:

```
1Ø PRINT RND (1)
```

Y ejecútelo, verá aparecer en el margen izquierdo de la pantalla un número del tipo Ø776433747. Hágalo varias veces y verá cada vez un número distinto, pero del mismo tipo. Trate de adivinar cuál será el próximo número. Falló, ¿verdad?

Añada al programa anterior la línea:

```
2Ø GO TO 1Ø
```

Y al ejecutarlo verá aparecer una ristra de números sin ninguna relación aparente entre ellos. Esto es generar números al azar.

Los números aparecen en la pantalla a gran velocidad y casi es imposible leerlos. Si desea ralentizar este movimiento, pulse la tecla CTRL, y mientras la mantenga pulsada verá aparecer los números más despacio. Si la suelta, volverá a adquirir velocidad. Pulse y suelte varias veces para practicar. (Esto es válido también para observar el listado de un programa.)

Ahora vamos a tratar de simular en el ordenador el lanzamiento de un dado. Teclee el programa siguiente:

```
1Ø REM ****************************
2Ø REM * LANZAMIENTO DE UN DADO *
3Ø REM ****************************
4Ø PRINT"[illegible]"
5Ø A=RND(1)
6Ø B=A*6
7Ø C=INT(B)
8Ø D=C+1
9Ø PRINT"[illegible]"
1ØØ PRINT TAB(19);D
11Ø PRINT:PRINT:PRINT:PRINT
12Ø PRINT"?OTRO LANZAMIENTO?(S/N)"
13Ø GET R$:IF R$="S" THEN 4Ø
14Ø IF R$<>"N" THEN 13Ø
15Ø PRINT:PRINT:PRINT TAB(15);"HASTA OTRA"
```

Cada vez que ejecute este programa obtendrá un número del 1 al 6 en el centro de la pantalla. Pero no conocemos de antemano cuál será el próximo número que aparecerá.

La explicación del programa anterior es muy sencilla. Queremos generar números enteros del 1 al 6, pero la función RND (1) genera números reales del Ø al 1 (nunca el Ø ni el 1), del tipo Ø847325437. ¿Cómo hacerlo? Bien, en la línea 5Ø introducimos en A el valor generado por RND (1). Si este número (com-

prendido entre el 0 y el 1) lo multiplicamos por 6, obtendremos un número entre el 0 y el 6 (nunca el 0 ni el 6). Por ejemplo, el 0.25 se convierte, al multiplicarlo por 6, en el 1.50, el 0.5 en el 3, el 0.75 en el 4.5, etc. En la línea 60 introducimos este valor en B. Pero un dado no genera números del tipo 2.754324375. Esto es, tenemos que conseguir números enteros entre el 1 y el 6. Para ello utilizamos en la línea 70 la función INT. Recuerde que INT toma la parte entera de un número. Así INT (2.7543243757)=2. De este modo, en la línea 70 introducimos en C un número entero del 0 al 5. Finalmente, lo único que necesitamos para conseguir nuestro dado es sumarle un 1 a C, y así el valor de D será un número entero entre el 1 y el 6, que es lo que imprimimos en la línea 100.

Las líneas 50, 60, 70 y 80 del programa anterior las podemos sustituir por la línea:

```
50 D=INT (RND(1)*6) + 1
```

La estructura general de la función es:

RND (X)

X: Argumento real. Su valor influye en la serie de números generados. En general, para un X dado, cada vez que encendemos el ordenador se generará la misma serie de números. Las excepciones son RND (0) y RND (TI), pues en ambos casos se utiliza el estado del reloj interno para determinar la serie a generar.

Si, por cualquier motivo, deseamos volver a utilizar la misma serie de números aleatorios hemos de introducir, antes de la generación, un RND con argumento negativo. Así, el programa:

```
10 REM *************
20 REM * ME REPITO *
30 REM *************
40 A=RND(-1)
50 FOR I=1 TO 5
60 PRINT I, RND(1)
70 NEXT I
```

imprimirá en la pantalla los mismos números *aleatorios* cada vez que lo ejecutemos. Para otro argumento negativo, por ejemplo, —2, la secuencia a repetir será distinta.

Finalmente, para terminar esta sección, vamos a diseñar un programa para jugar a los dados contra el ordenador. Empezaremos con una cantidad de dinero, pongamos 20 pesos, y después el ordenador nos pedirá que apostemos a un número del 1 al 6. Tras aceptar nuestra apuesta, generará al azar un número del 1 al 6 y nos dirá cuál ha sido nuestra suerte.

```
10 REM ******************
20 REM * JUEGO DEL DADO *
30 REM ******************
40 CAPITAL=100
50 PRINT"□"
60 PRINT:PRINT:PRINT TAB(10);"DISPONE "+
   "DE";CAPITAL;"PTS"
70 PRINT:PRINT TAB(10);
   "?CUANTO APUESTA";
80 INPUT APUES
90 IF APUES<0 OR APUES>CAPITAL
   THEN 50
100 PRINT:PRINT TAB(6);"?A QUE NUMERO?";
105 GET A$: IF A$=""THEN 105
110 IF ASC(A$)<49 OR ASC(A$)>54 THEN 105
120 A=VAL(A$):PRINT A
130 REM ****************************
140 REM * M=VUELTAS QUE DA EL DADO *
150 REM ****************************
160 M=INT(RND(1)*100)
165 PRINT:PRINT:PRINT:PRINT:PRINT
170 FOR I=1 TO M
180 D=INT(RND(1)*6)+1
190 FOR J=1 TO 500/(M-I+1):NEXT J
200 PRINT"□";TAB(20);D
210 NEXT I
220 PRINT:PRINT:PRINT
230 IF D=A THEN PRINT TAB(6);"HA GANADO"
   ;APUES*5;"PTS":G=1:P=0
240 IF D<>A THEN PRINT TAB(6);"HA "+
   "PERDIDO";APUES;"PTS":G=0:P=1
250 CAPITAL=CAPITAL+(G*5-P)*APUES
260 PRINT:PRINT:PRINT
270 IF CAPITAL=0 THEN PRINT TAB(8);"SE "
   +"HA ARRUINADO":GO TO 310
280 PRINT"AHORA TIENE";CAPITAL;"PTS.";
   "OTRA APUESTA(S/N)?"
290 GET R$:IF R$="S" THEN 50
300 IF R$<>"N" THEN 290
310 END
```

La línea 90 se encarga de comprobar que no se puede apostar una cantidad negativa y que no juege más dinero del que tiene. De que no engañe al ordenador, apostando a un número distinto de los del dado o a una letra, se encarga la línea 110. Pero quizá lo más curioso de este programa se desarrolla en las líneas 160-210. Con ellas tratamos de simular en pantalla el lanzamiento de un dado en una forma muy sencilla. Seguro que usted puede mejorarlo. En la línea

160 se genera al azar el número de vueltas que dará el dado, las primeras vueltas son más rápidas y poco a poco el movimiento se ralentiza hasta que el dado se para. La línea 190 trata de llevar a cabo este efecto. Por ejemplo, suponga que hemos obtenido M=50. Así, para la primera vuelta, I=1, el «valor final», del bucle FOR que genera la pausa, es de 500/(50+1—1)=10, mientras que para la última vuelta, I=50, este valor es de 500 /(50—50+1)= 500. En el resto del programa se realiza la impresión de los resultados y se pregunta si se desea realizar otra apuesta. Fíjese en la forma de calcular el nuevo capital. Según si ha acertado o no, las variables G (ganar) y P (perder) toman los valores G=1 y P=0 ó G=0 Y P=1. De este modo, en la línea 250 se utilizan estos valores para calcular el nuevo capital a partir del anterior y de la cantidad apostada. Realmente, hubiera sido más sencillo prescindir de las variables G y P y utilizar una línea 240, tal como:

240 CAPITAL=CAPITAL+(—(A=D)*5+(A< >D))*APUESTA

Recuerde que si una relación es cierta, el ordenador la evalúa como un —1, y si es falsa, la evalúa como un 0. Observe que si acierta, la máquina le premia con una cantidad igual a cinco veces su apuesta. ¡Animo!

FICHEROS DE DATOS

El objetivo principal de la informática es la manipulación de grandes cantidades de datos. Por ejemplo, en una escuela habrá que manejar los números de matrículas de los alumnos, sus nombres y direcciones, edad, historial académico, etc. Hasta ahora hemos dedicado nuestro esfuerzo a aprender las estructuras de que disponemos para constituir programas e incluso ya sabemos cómo crear ficheros de programas en la Datassette. De este modo iremos diseñando programas y grabándolos en la cinta para volver a utilizarlos cuando los necesitemos. Vamos a dedicar esta sección a introducir estructuras avanzadas para el manejo de datos y para estudiar cómo crear ficheros de datos, nombres, direcciones, etc., en la Datassette,

Variables con índices (listas y tablas)

Suponga que necesitamos diseñar un programa para introducir las notas de los alumnos de una clase, pero queremos que cada nota esté en una variable distinta, no acumularlas en una sola variable, para luego sacar la media como hacíamos en el capítulo anterior. Evidentemente, si utilizamos las variables que ya conocemos nos quedará un programa terriblemente complicado y engorroso. Pruebe a hacerlo. Afortunadamente el lenguaje BASIC nos proporciona otro método para hacerlo: las variables con índices (ver Fig. 3.3). Intuitivamente, una variable con un índice es como un conjunto de casillas contiguas en cada una de las cuales tenemos guardado, o podemos guardar, un dato. Así, fijándonos en la figura 3.3,

nombre de variable: A	15	43	3.7	85.4	....	
valor del índice: I →	1	2	3	4		

Fig. 3.3. Concepto intuitivo de variable con índice.

si le decimos al ordenador PRINT A(1), saldrá un 15 en la pantalla, pues éste es el contenido de la primera casilla de A, y si le decimos PRINT A(4), imprimirá el número 85.4.

Claro que, en general, antes de utilizar una variable con un índice (las variables con un solo índice se suelen llamar listas), tenemos que indicarle al ordenador de cuántas casillas va a constar para que pueda reservar espacio de memoria para contenerlas. Esto lo conseguimos con el comando DIM. Por ejemplo, DIM A(50) nos permite utilizar la variable A(I) con I variando hasta 50. Como siempre, el mejor método de entenderlo es con un ejemplo:

```
10 REM ***********************
20 REM * NOTAS DE UNA CLASE *
30 REM ***********************
40 PRINT"♥"
50 PRINT:PRINT:PRINT
   "CUANTOS ALUMNOS TIENE";
60 INPUT N
70 REM *****************************
80 REM * INTRODUCCION DE LAS NOTAS *
90 REM *****************************
100 DIM A(N)
110 FOR I=1 TO N
120 PRINT:PRINT"NOTA DEL";I;"ALUMNO";
130 INPUT A(I)
140 NEXT I
150 REM ****************************
160 REM * CALCULO DE LA NOTA MEDIA *
170 REM ****************************
180 FOR I=1 TO N
190 SUMA =SUMA+A(I)
200 NEXT I
210 PRINT:PRINT:PRINT
220 PRINT"LA NOTA MEDIA DE LOS";N;
    "ALUMNOS ES";SUMA/N
```

Observe que con una sola variable A(I) podemos introducir y guardar las notas de 40, 50 ó 1000 alumnos. Además, el programa vale para cualquier tamaño de clase, pues lo primero que hace (línea 50) es preguntarle cuántos alumnos tiene en la clase, N, y tras recibir su respuesta, la línea 100 reserva espacio en la memoria DIM A(N) para poder almacenar las notas de todos sus alumnos. Las líneas 110-140 se ocupan de introducir sucesivamente las notas de los alumnos, indicándole en cada momento el número del alumno al que se está refiriendo. Fíjese qué fácil y qué sencillo y trate de imaginarse lo que sería esto si utilizáramos variables sin índice.

Ahora ya tenemos las notas guardadas en la memoria de la máquina y podemos hacer con ellas lo que queramos: sacar un listado de las notas, calcular la media, etc. Como ejemplo, en el programa anterior las líneas 180-220 se encargan de leer las notas guardadas en A(I) y calcular e imprimir la media.

La estructura general de la sentencia DIM para listas es (DIM viene de palabra dimensión).

DIM N(I)

I: Es un argumento numérico que bien puede ser una constante, una variable o una expresión cuyo resultado sea numérico.
N: Nombre de variable tiene las mismas reglas de construcción que las variables sin índices y pueden ser bien de variable real, por ejemplo N, de variable entera, N%, o de variable alfanumérica, N$.

Para practicar un poco más vamos a escribir un programa que pida l nombres y las notas de las evaluaciones de los alumnos. Al final calculará nota media de cada alumno e imprimirá en la pantalla el nombre del alumno su nota final. Utilizaremos las variables N$() para los nombres, la A1() pa las notas de la primera evaluación, A2() para las de la segunda y A3() para l de la tercera. Al final introduciremos la nota final de cada alumno en F() pasaremos a imprimir los resultados.

```
10 REM ********************
20 REM * MAS NOTAS MEDIAS *
30 REM ********************
40 PRINT"J"
50 INPUT"NUMERO DE ALUMNOS";N
60 DIM N$(N),A1(N),A2(N),A3(N),F(N)
70 FOR I=1 TO N
75 PRINT"J"
80 PRINT"NOMBRE Y APELLIDOS DEL";I;
   "ALUMNO"
90 INPUT N$(I)
100 PRINT:PRINT"NOTA DE EVALUACION 1";
110 INPUT A1(I)
120 PRINT:PRINT"NOTA DE EVALUACION 2";
130 INPUT A2(I)
140 PRINT:PRINT"NOTA DE EVALUACION 3";
150 INPUT A3(I)
160 F(I)=(A1(I)+A2(I)+A3(I))/3
170 NEXT I
180 PRINT"J"
190 REM *************************
200 REM * IMPRESION EN PANTALLA *
210 REM *************************
220 FOR I=1 TO N
230 PRINT:PRINT"ALUMNO:";N$(I)
240 PRINT"NOTA FINAL";F(I)
250 NEXT I
```

Observe cómo en la línea 60 realizamos varios dimensionamientos y fíjese en que están separados por comas. El resto del programa tiene poco más que comentar. Quizá sería interesante introducir una variante para, tras escribir unos cuantos nombres, salir del primer bucle FOR y sacar un listado de los nombres ya almacenados. Pero recuerde que si en un programa ya ha introducido algunos nombres y se interrumpe la ejecución, por ejemplo, pulsando RUN/STOP y RESTORE, si vuelve a ejecutar el programa RUN, borrará todos los datos que tenga introducidos.

Claro que si antes manifestábamos la necesidad de utilizar variables con un índice, debido al engorro que sería utilizar una variable para cada dato, ahora parece lógico esperar que si, además de tener una lista de clientes, tenemos una serie de datos para cada cliente, dispongamos también de alguna estructura para almacenar estos datos de una forma lógica. De aquí surge el concepto de variable con dos índices o tabla (ver la Fig. 3.4).

	1	2	3		M → J
1	Manuel	Gómez	López	......	4
2	José	Garrido	Fernández	......	7
3	Javier	Pérez	Bermúdez	......	6
	.	.	.	.	.
N ↓ I	Antonio	Esteban	Gómez	......	5

Fig. 3.4. Concepto intuitivo de variable con dos índices.

En cada fila tenemos a un cliente (en este caso alumno), y a esto se le llama ficha. Cada ficha tiene varios campos, tantos como indiquemos con el índice J y en cada campo guardamos un dato sobre el cliente. Así, en el campo 1 (J=1) de la primera ficha (I=1) tenemos guardado el nombre (Manuel) del primer alumno, y en el campo M (M tendrá cierto valor) de cada ficha tenemos guardada la nota final de la clase. Recuerde que I se refiere a la fila y J al campo.

Con este nuevo bagaje vamos a atacar otra vez el ejemplo anterior. Tendremos un fichero N$(I,J), donde I es el número de alumnos y J es el número de campos de cada ficha. Pongamos, para este ejemplo, que tenemos 7 campos «NOMBRE», «1. APELLIDO», «2. APELLIDO», «1. EVALUACION», «2. EVALUACION», «3. EVALUACION», y «NOTA FINAL». Estos son los títulos que vamos a poner a los campos y es muy importante para conocer en cada momento qué dato estamos tecleando. Compare su solución con la siguiente:

```
10 REM **********************
20 REM * FICHERO DE ALUMNOS *
30 REM **********************
40 PRINT"J"
50 INPUT"NUMERO DE ALUMNOS";N
```

```
60 DIM N$(N,7),T$(7)
70 T$(1)="NOMBRE":T$(2)="1.APELLIDO":
   T$(3)="2.APELLIDO"
75 T$(4)="1.EVALUACION":
   T$(5)="2.EVALUACION"
80 T$(6)="3.EVALUACION":
   T$(7)="NOTA FINAL"
90 FOR I=1 TO N
95 PRINT"♥"
100 FOR J=1 TO 3
110 PRINT:PRINT T$(J)+" DEL";I;"ALUMNO";
120 INPUT N$(I,J)
130 NEXT J
140 PRINT"♥"
150 PRINT:PRINT"ALUMNO:"+N$(I,1)+" "+
    N$(I,2)+" "+N$(I,3)
160 FOR J=4 TO 6
170 PRINT:PRINT TAB(8);T$(J);
180 INPUT N$(I,J)
190 NEXT J
200 N$(I,7)=STR$((VAL(N$(I,4))+
    VAL(N$(I,5))+VAL(N$(I,6)))/3)
210 NEXT I
220 REM ******************************
230 REM * IMPRESION DE NOTAS FINALES *
240 REM ******************************
260 FOR I=1 TO N
265 PRINT"♥"
270 FOR J=1 TO 3
280 PRINT N$(I,J);" ";
290 NEXT J
300 PRINT:PRINT TAB(8);T$(7)+"="+N$(I,7)
310 PRINT:PRINT:PRINT:PRINT
320 PRINT"DESEA PASAR AL ALUMNO "+
    "SIGUIENTE(S/N)?"
330 GET R$
340 IF R$="N" THEN 399
350 IF R$<>"S"THEN 330
360 NEXT I
399 END
```

Esto ya es algo más complicado. Para almacenar todos los datos, ya sean non bres o notas, hemos utilizado una tabla alfanumérica N$(I,J). Los títulos de lo campos los hemos guardado en una lista alfanumérica T$(J). Evidentemen los podríamos haber guardado en variables alfanuméricas separadas, pero est imposibilitaría la sencillez de los bucles que utilizamos posteriormente. Lo qu sí hubiera sido importante es no asignar los títulos de los campos dentro del pr grama (líneas 7Ø y 8Ø), sino utilizar un bucle FOR y un INPUT para pedir al usuario que introduzca los títulos que quiera; pruebe a realizar esta modificación. El resto del programa se dedica a realizar la introducción de los datos y a la impresión de los resultados finales. Estudie el programa con detenimiento, pues hay muchos detalles que podrá utilizar en sus propios diseños. En la línea 32Ø introducimos una nueva variante. En esta parte del programa, línea 22Ø-399, se realiza la impresión de los resultados, y en lugar de dejar que aparezcan uno detrás de otro, sin posibilidad de leerlos (aunque sabemos que pulsando la tecla CTRL la velocidad se ralentiza), le preguntamos al usuario si desea ver el alumno siguiente; si no es así, la lína 34Ø se encarga de sacarnos del bucle, y, en caso contrario, se borran de la pantalla los datos del alumno anterior y se imprimen los del siguiente.

Una variable puede tener más de dos índices; por ejemplo, podríamos utilizar una variable numérica con tres índices N(I, J, K), para almacenar posiciones en un espacio tridimensional. Es decir, esto aún es imaginable, pero es mucho más difícil tratar de imaginarse el funcionamiento de variables con cuatro, cinco o más índices. Por tanto, podríamos resumir que la estructura general de la sentencia DIM es:

DIM N(I, J, K, ...)

> I, J, K, ..., son los índices y han de ser valores numéricos: constantes, variables o expresiones aritméticas.
> N es el nombre de la variable y sigue las mismas reglas que en el caso de variables normales. Esto es, para variables enteras hay que terminar el nombre con el símbolo %, y con variables alfanuméricas, con el símbolo $.

Además es bueno recordar los puntos siguientes:

1. Si realizamos DIM A(M), en realidad se reservan M+1 espacios para A, pues también se puede utilizar A(Ø). Sin embargo, no es frecuente utilizar A(Ø) en los programas.
2. Si pasamos, dentro de la ejecución de un programa, dos veces por el mismo DIM, obtendremos un mensaje de error. No podemos dimensionar algo que ya está dimensionado.
3. Si dimensionamos la variable DIM A (5Ø) y después nos equivocamos y tratamos de utilizar, por ejemplo, A (54), obtendremos un mensaje de error.
4. Podemos dimensionar con un número no entero, pero el C-64 tomará la parte entera antes de dimensionar, esto es, DIM A (16.7) equivale a DIM A (16).
5. Variables con menos de cuatro índices, por ejemplo, A (I, J), en las que los índices no vayan a tomar valores mayores de 1Ø, esto es, Ø ≤ I ≤ 1Ø, Ø ≤ J ≤ 1Ø, no necesitan ser dimensionadas. El C-64 reserva espacio para estos datos en tanto que intentemos utilizarlas.

El programa anterior es un ejemplo de introducción de datos en un fichero, que será de utilidad para cualquier tipo de base de datos que queramos construir, por ejemplo, clientes, el estado de cuentas; productos en el almacén, cantidad, precios, entradas y salidas diarias; colección de discos, tipo de música, autor, título, etc. Sin embargo, en muchas ocasiones más que sacar un listado lo que deseamos es buscar datos de una sola ficha. Por ejemplo, refiriéndose al programa anterior, deseamos buscar las notas parciales y final de cierto individuo, para ello podíamos sustituir las líneas que se ocupan de sacar el listado por las siguientes:

```
1000 REM*********************************
1010 REM      LISTADO SELECTIVO         *
1020 REM*********************************
1030 PRINT"□"
1040 PRINT:PRINT"DESEA VER ALGUN ALUMNO (S/N)?'
1050 GET R$:IF R$="N" THEN 1999
1060 IF R$<>"S" THEN 1050
1070 PRINT"□"
1080 PRINT:PRINT:PRINT"NOMBRE";:INPUT N$
1090 PRINT:PRINT"1.APELLIDO";:INPUT A1$
1100 PRINT:PRINT"2.APELLIDO";:INPUT A2$
1110 FOR I=1 TO N
1120 IF N$(I,1)=N$ AND N$(I,2)=A1$ AND
     N$(I,3)=A2$ THEN 1150
1130 NEXT I
1140 PRINT:PRINT:PRINT"NO ENCUENTRO NINGUN
     ALUMNO CON ESE  NOMBRE":GOTO 1040
1150 PRINT:PRINT:PRINT LEFT$(T$(4),9),LEFT$
     (T$(5),9) ,LEFT$(T$(6),9),LEFT$(T $(7),9)
1160 PRINT:PRINTN$(I,4),N$(I,5),N$(I,6),N$(I,7)
1170 GOTO 1040
1999 END
```

Esta es una rutina fácilmente manejable y adaptable a cualquier caso p cular. En las líneas 1080-1100 se pregunta el nombre y apellidos de la per buscada, y las líneas 1110-1130 se encargan de realizar la búsqueda. Si en chero no existe ninguna persona con estos datos, el ordenador le informa (l 1140) y cede el control a la línea 1040 por si desea buscar a otra persona. búsqueda tiene éxito, las líneas 1150 y 1160 se encargan de imprimir los c en forma ordenada. Recuerde que la coma divide a la pantalla en cuatro z de impresión cada una con 10 caracteres, por ello empleamos LEFT$ (T 9), no vaya a ser que tengamos títulos demasiado largos que nos estropee presentación de los resultados. Otra norma puede ser la de poner al prin del programa, en la introducción, unos controles para que tanto los títul los campos como su contenido no sobrepasen unos límites prefijados.

Fíjese en que en el mismo programa hemos utilizado la variable con ín N$(I,J) y la variable normal N$. El Commodore le permite esto y distingue fectamente entre ambas variables. En las líneas 1080-1100, antes del símbol dos puntos que separa el PRINT del INPUT, hemos puesto el símbolo de p to y coma «;», para que el cursor del INPUT aparezca justo a continua de la pregunta formulada con el PRINT.

Y para terminar de practicar con las variables con índices vamos a prese un algoritmo que será casi imprescindible en un programa de gestión de d Un algoritmo de ordenación. Hay diversos algoritmos de ordenación, pero no hay lugar para exponerlos todos o para medir sus ventajas o desvent Supongamos que tenemos un fichero A( ) de números que deseamos ordena menor a mayor e imaginemos el peor de los casos: están justo al revés.

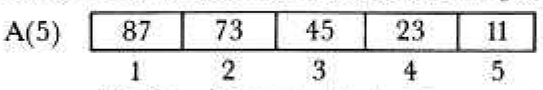


Fig. 3.5. Lista ordenada al revés.

Empecemos por comparar los lugares 1 y 2; si el número que hay en 1 es menor que el que hay en 2, ya están ordenados. En caso contrario procederemos primero a cambiarlos y después pasaremos a comprobar si los números que están en los lugares 2 y 3 están ordenados. Observe que si en la primera comparación hemos intercambiado los números A(1) y A(2), en la segunda comparación, entre A(2) y A(3), el número que hay en A(2) es el que antes estaba en A(1). Para intercambiar los números se utiliza una variable auxiliar, que en este caso llamaremos CAMBIO, que sólo sirve para esto, para ayudar en el cambio

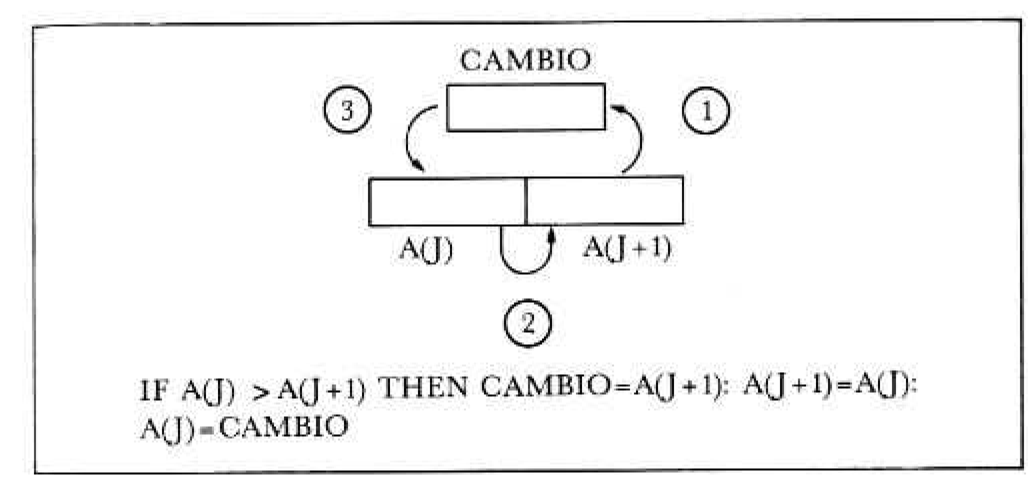


Fig. 3.6. Pasos para intercambiar el contenido de las casillas.

En la figura siguiente se exponen todos los pasos necesarios para la ordenación. Primero comparamos las casillas A(1) y A(2), si hace falta se realiza el cambio, y después pasamos a comparar las casillas A(2) y A(3), y así sucesivamente hasta comparar la A(4) con la A(5) y realizar el cambio, si es necesario. Observe que, tras la primera vuelta, estamos seguros de que el mayor número, en este caso el 87, ocupa la última casilla.

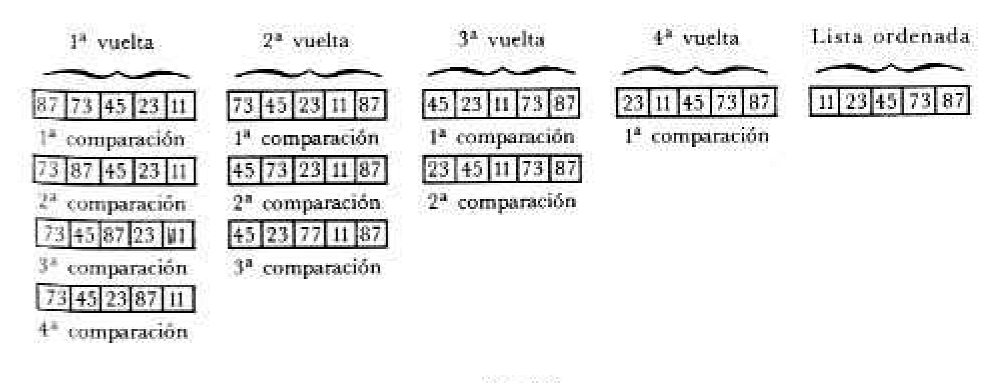


Fig. 3.7.

Tras esto empezamos la segunda vuelta y comparamos la A(1) con la A(2), la A(2) con la A(3), la A(3) con la A(4), y en este caso ya no hace falta comparar la A(4) con la A(5), pues en la primera vuelta pusimos el mayor en la A(5) y en esta segunda vuelta hemos colocado el segundo mayor en la A(4). Tras esto se concluye que en cada vuelta hay que hacer una comparación menos, pues sabemos que por lo menos tenemos ordenado un número más. Así, si el número de elementos a ordenar es N y estamos en la vuelta I, el número de comparación necesario es N-I.

Núm. de elementos n	*Núm. de vueltas i*	*Núm. de comparaciones n-i*
5	1	4
5	2	3
5	3	2
5	4	1

Fig. 3.8. Relación entre número de vueltas y número de comparaciones.

Todo esto lo podríamos expresar en el programa:

```
10 REM**********************************
20 REM*    ORDENACION DE LISTAS        *
30 REM**********************************
40 PRINT"♥"
50 REM**********************************
60 REM*   INTRODUCCION DE NUMEROS      *
70 REM**********************************
80 PRINT"CUANTOS NUMEROS VA A INTRODUCIR";
90 INPUT N: DIM A(N)
100 FOR I=1 TO N
105 PRINT"♥"
110 PRINT"TECLEE EL";I;"NUMERO"
120 INPUT A(I)
130 NEXT  I
140 PRINT"LOS NUMEROS INTRODUCIDOS SON"
150 FOR I=1 TO N
160 PRINT"A(";I;")=";A(I)
170 NEXT I
180 PRINT"SI DESEA VERLOS ORDENADOS"
190 PRINT"PULSE CUALQUIER TECLA"
200 GETR$:IFR$=""ORR$=CHR$(13)THEN 200
210 REM**********************************
220 REM**       ORDENACION            **
230 REM**********************************
240 FOR I=1 TO N-1
250 FOR J=1 TO N-I
260 IF A(J)>A(J+1) THEN CAMBIO=A(J+1):
    A(J+1)=A(J):A(J)=CAMBIO
270 NEXT J
280 NEXT I
290 REM**********************************
300 REM*       LISTA ORDENADA          *
310 REM**********************************
320 PRINT"♥"
330 PRINT"LOS NUMEROS ORDENADOS SON"
340 FOR I=1 TO N
350 PRINT"A(";I;")=";A(I)
360 NEXT I
```

En la primera parte, líneas 10-170 se realiza la introducción y visualización de los números a ordenar. Esta labor de introducción es muy pesada, así que lo mejor, una vez introducidos y ordenados los datos, es grabarlos en algún dispositivo de almacenamiento, cinta o disquete, como veremos posteriormente. En las líneas 180-200 se le pregunta si desea verlos ordenados. En cuanto pulse una tecla comenzará la ordenación. Claro que si no desea verlos ordenados, no tiene más remedio que pulsar RUN/STOP y RESTORE para interrumpir la ejecución del programa. Trate de arreglar esto. De la ordenación se ocupan las líneas 240-280. En la línea 240 el índice I varía desde 1 a N-1. Como vimos en la figura 3.8, si tenemos N datos a ordenar, tenemos que dar N-1 vueltas. En el ejemplo de la figura 3.8 teníamos cinco datos y, por tanto, el número de vueltas era cuatro. La línea 250 se encarga de realizar el número de comparaciones necesarias en cada vuelta. Como hemos visto en las figuras, en la vuelta I tenemos que realizar N-I comparaciones. Las líneas 290-360 se ocupan de la impresión de los datos ordenados, y si ha introducido muchos números, recuerde que para ralentizar el listado debe de pulsar la tecla CTRL.

Otro problema es el de ordenar listas alfabéticas. Por ejemplo, ordenar alfabéticamente los nombres de una clase. Parace más complicado, pero para el ordenador es lo mismo. En la línea 290 lo único que cambiaría serían los nombres de variables, que tendrían que ser alfanuméricas. Cuando el ordenador compara dos apellidos, por ejemplo:

GARCIA
GARCES

va comparando letra a letra. Recuerde que las letras tienen un número de código, el de la A es el 65, el de la B el 66, etc. El ordenador lo que compara es el número de código, y por ello, en el ejemplo de GARCIA y GARCES, como la primera letra coincide se pasa a la segunda y luego a la tercera, cuarta, y al llegar a la quinta, la E es menor que la I (refiriéndonos a sus números de código), y por ello sabemos que GARCES ha de ir delante de GARCIA. Pruebe a crear un ejemplo para la ordenación alfabética y recuerde que hay que comparar primero el primer apellido, después el segundo y, finalmente, el nombre. ¡Suerte!

Ficheros internos de un programa: READ, DATA, RESTORE

A veces, necesitamos mantener cierta información fija en el programa y tener que asignarla mediante LET o introducirla mediante INPUT; puede ser

demasiado lento. Para esto disponemos de una sentencia que nos permite man tener datos en una línea de programa: la sentencia DATA. En una línea DAT podemos almacenar constantes numéricas o alfanuméricas, separadas por co mas. La sentencia DATA no es ejecutable, es decir, cuando el ordenador la lee sabe dónde está, pero no tiene que hacer nada con ella hasta que no le pidamos en otra parte del programa, que la lea. Podemos tener tantas líneas DATA com queramos, y es una buena costumbre el acumularlas al principio o al final de programa.

Para leer el contenido de una línea DATA tenemos que utilizar la sentenci READ. La estructura general de la sentencia READ es:

READ «lista»

Lista: es una lista de nombres de variables reales, enteras o alfanuméricas, separadas por comas. El tipo de variable en la sentencia READ debe de coincidir con el tipo de dato en la sentencia DATA.

```
1Ø REM EJEMPLO
2Ø DATA 3, 5, 6, "HOLA"
3Ø READ A,B,C,A$
4Ø PRINT A,B,C,A$
```

Al ejecutar este programa, la línea 3Ø leerá de la línea DATA los valores 3 5, 6 y «HOLA» y los introducirá, respectivamente, en las variables A, B, C y A$. Después, la línea 4Ø se encargará de la impresión del contenido de esta variables en la pantalla. Compruebe que si a la línea DATA le cambia el núme ro de línea, por ejemplo, de 2Ø a 1ØØ, el resultado es el mismo.

Como ha observado en el ejemplo, READ debe ir seguido de los nombre de variable numérica o alfanumérica que se correspondan con el dato que h de leer. READ siempre lee el primer dato, del bloque de líneas DATA, que aú no haya sido leído. Es decir, el ordenador lleva en su memoria un puntero qu le indica cuál es el siguiente dato que ha de leer. Cuando nos encontramos co el primer READ, se busca el primer DATA que haya y se coloca la direcció del primer dato como contenido del puntero. Tras leer un dato, el contenid del puntero aumenta en una dirección para que así la próxima vez leamos e dato siguiente. Si tratamos de leer más datos de los que hay obtendremos u mensaje de error.

```
1Ø REM**********************************
2Ø REM*      NO HAY TANTOS DATOS       *
3Ø REM**********************************
4Ø DATA 1,2,3,4
5Ø DATA 5,6,7,8
6Ø READ A,B,C,D
7Ø PRINTA,B,C,D
8Ø GO TO 6Ø
```

Cuando el control llega por primera vez a la línea 6Ø, se busca la dirección del primer DATA y se van leyendo sucesivamente los valores de A, B, C y D. Después, la línea 7Ø se encarga de imprimirlos y la 8Ø devuelve el control a la línea 6Ø, que ha de leer otros cuatro datos. Ahora el puntero marca la dirección del 5 de la línea 5Ø, y por ello se leerán e imprimirán los números 5, 6, 7 y 8 de la línea 5Ø. Después el control vuelve a la línea 6Ø, pero ya no hay más datos para leer y por ello obtendremos un mensaje de error: OUT OF DATA ERROR IN 6Ø.

Claro que a veces nos interesará leer cierto número de veces el mismo dato, y para ello podemos utilizar el comando RESTORE. La estructura general del comando RESTORE es bastante sencilla

RESTORE

y su acción es la de hacer que la próxima dirección de lectura vuelva a ser la del primer DATA que tengamos:

```
1Ø REM**********************************
2Ø REM*      UTILIZANDO RESTORE        *
3Ø REM**********************************
4Ø DATA 1,2,3,4,5,6,7,8
5Ø READ A
6Ø PRINT A
7Ø RESTORE
8Ø GO TO 5Ø
```

Con este programa siempre estaremos leyendo e imprimiento el 1 de la línea 4Ø, pues el comando RESTORE obliga a que la próxima dirección de lectura vuelva a ser la primera.

La estructura general de la línea DATA es:

DATA «lista»

Lista: es una serie de datos numéricos o alfanuméricos separados por comas. Los datos alfanuméricos no es necesario encerrarlos entre comillas. Si tratamos de leer un dato alfanumérico con un READ numérico obtendremos un mensaje de error.
Si leemos mediante un READ A% un dato numérico no entero se tomará la parte entera del número correspondiente.

Vamos a diseñar un programa «Diccionario de Idiomas». Su funcionamiento ha de ser el siguiente: tras preguntarnos qué día de la semana deseamos (del 1 al 7) y en qué lenguaje lo deseamos (español, inglés, o francés), nos presentará en la pantalla el número del día que hemos pedido y cómo se escribe en el idioma elegido.

```
10 REM**********************************
20 REM*   DICCIONARIO DE IDIOMAS       *
30 REM**********************************
40 PRINT"ℷ"
50 PRINT:PRINT TAB(12);"DIA DE LA SEMANA"
60 PRINT:PRINT:PRINT
70 PRINT"PULSE UN NUMERO DEL 1 AL 7"
80 GET A$: IF A$="" THEN 80
85 IF ASC(A$)<49 OR ASC(A$)>55 THEN 80
90 A=VAL(A$)
100 PRINT"ℷ"
110 PRINT:PRINT:PRINT"LO DESEA EN": PRINT
120 PRINT TAB(8);"1.ESPANOL"
130 PRINT TAB(8);"2.INGLES"
140 PRINT TAB(8);"3.FRANCES"
150 PRINT:PRINT:PRINT
160 PRINT"PULSE UN NUMERO DEL 1 AL 3"
170 GET R$: IF R$="" OR R$<"1" OR R$>"3"
    THEN 170
180 R=VAL (R$)
190 RESTORE
200 REM**********************************
210 REM*       BUSQUEDA DE DATO         *
220 REM**********************************
230 FOR I=1 TO (R-1)*7+A
240 READ D$
250 NEXT I
260 PRINT"ℷ"
270 PRINT"EL DIA";A;"SE ESCRIBE ";D$
280 PRINT
290 PRINT
300 PRINT"DESEA BUSCAR OTRO DIA (S/N)"
310 GET K$: IF K$="S" THEN 40
320 IF K$<>"N" THEN 310
330 PRINT:PRINT TAB(15);"HASTA OTRA"
340 END
350 DATA LUNES,MARTES,MIERCOLES,JUEVES,
    VIERNES,SABADO,DOMINGO
360 DATA MONDAY,TUESDAY,WEDNESDAY,
    THURSDAY,FRIDAY,SATURDAY,SUNDAY
370 DATA LUNDI,MARDI,MERCREDI,JEUDI,
    VENDREDI,SAMEDI,DIMANCHE
```

La única novedad está en cómo se utilizan las variables R y A para calcu el valor final del bucle FOR (línea 23Ø) que se encarga de ir leyendo los DA uno tras otro. Los días de la semana son siete, y, por ejemplo, si queremos l el cuarto día en inglés (R=2, A=4), entonces (R−1)∗7+A=11, así leeremos o datos, los siete en español y los cuatro primeros en inglés, de modo que el último dato que leeremos con D$ (línea 24Ø) será «THURSDAY». Aumente su diccionario y juegue con sus amigos. Cambie este programa y almacene los datos en diversas variables con índices.

Ficheros en la Datassette

Es una pena pasarse una hora o más introduciendo datos, en algunos de los ejemplos anteriores, y saber que después, cuando apaguemos el Commodore, se borrarán estos datos de su memoria.

En el capítulo primero vimos cómo guardar los programas en la Datassette y crear así ficheros de programas que podemos conservar en cintas después volver a reproducirlos directamente desde las cintas sin necesidad de teclearlos. Algo similar tendremos que hacer con los datos. Parece obvio. Es muy confortable poder grabar los datos, nombres, direcciones, etc., y después poder reproducirlos para leerlos, modificarlos o lo que queramos.

Estos almacenes de datos en dispositivos periféricos, cintas, disquetes, reciben el nombre de ficheros de datos, y ahora vamos a aprender cómo crearlos y utilizarlos con la Datassette. En el apéndice «Otros periféricos» se explica cómo hacerlo con el dispositivo de discos. La evaluación entre si comprar una Datassette o un dispositivo de discos es muy sencilla: la Datassette es mucho más barata, pero el dispositivo de discos es muchísimo más rápido. Con lo que todo depende de cómo evalúe su propio tiempo.

Sea cual sea el dispositivo que elija, sea precavido y trate las cintas o discos con cierta precaución. No las exponga a fuentes de calor, magnetismo o humedad y, por si acaso se le olvidan las precauciones anteriores, realice copias de seguridad (copie los datos o programas en varias cintas o discos).

Existen varios tipos de ficheros de datos, pero con la Datassette estamos limitados a utilizar un solo tipo: *los ficheros de acceso secuencial.* Un fichero de acceso secuencial tiene la característica siguiente:

Para acceder a un dato hay que pasar primero por todos los datos que le preceden.

Las instrucciones básicas para manejar ficheros en el C-64 son: OPEN, PRINT#, INPUT#, GET# y CLOSE. Vamos a analizarlas una a una.

OPEN nf, nd, ds, f$

- nf: Es el número del fichero. El número que utilizamos para tener acceso a este fichero. Podemos elegir cualquier número entre Ø y 225.
- nd: Es el número del dispositivo. La Datassette es el dispositivo 1. Si trabajamos con una sola unidad de discos utilizaríamos el número 8, etc.
- ds: Es un número que indica para qué hemos abierto el fichero. Los valores que puede tener dependen del dispositivo que utilicemos. En la Datassette son:
  - Ø: Abrir fichero para leer.
  - 1: Abrir fichero para escribir.
  - 2: Abrir fichero para escribir y añadir una marca de fin de tabla (para avisarnos de cuándo termina el fichero).
- f$: Nombre que le damos al fichero, en general una expresión o variable alfanumérica. Como máximo puede tener 16 caracteres.

Trabajando con Datassette podemos tener abiertos simultáneamente hast: 10 ficheros; esto es, 10 números de ficheros «nf». Pero si en una misma cint: abrimos un fichero y a continuación otro, en la zona que deberíamos guarda para los datos del primer fichero grabamos la cabecera del segundo y despué iremos mezclando los datos de uno y otro, con lo cual será muy difícil recupe rarlos. El método más aconsejable es utilizar varias cintas, una por cada ficher que deseemos abrir. Ya le advertí que era pesado.

No es obligatorio especificar el valor de «ds». Si no ponemos nada, el orde nador le asigna un cero. También podemos prescindir de indicar el nombre de fichero «f$». En este caso, si estamos leyendo, tomaremos el primer fichero qu encuentre la Datassette, y si estamos escribiendo, le pondremos un nombre va cío al fichero.

Cuidado, pues podemos prescindir de «ds» y de «f$», pero lo que no nos de jará hacer es darle valor a «f$» y no a «ds». La situación inversa sí está permitida

PRINT# nf, «lista a escribir»

nf: Número del fichero al que ya nos referíamos en la instrucción OPEN.
«Lista a escribir»: Dato o nombre de variable cuyo contenido se va a imprimir en el fichero.

Para poder utilizar esta instrucción es necesario que previamente hayamo abierto (OPEN) el fichero número «nf». No podemos dejar un espacio en blan co entre el PRINT y #, pues obtendremos un mensaje de error. Además, co PRINT# no podemos utilizar ni TAB ni SPC.

Como veremos en el apéndice B, si en lugar de PRINT escribimos un sím bolo de interrogación «?», el ordenador lo interpreta como un PRINT. Pue bien, esto no es válido en el caso de PRINT#, y si lo utilizamos nos dará erro

Si la «lista a escribir» no termina en coma o en punto y coma, al final d estos datos se grabará el carácter correspondiente a CHR$ (13), es decir, un ca rácter que representa un salto de línea.

CLOSE, nf

Tras operar con cada fichero es necesario cerrarlo. Si tras abrirlo (OPEN para escribir, no lo cerramos (CLOSE), después no podremos abrirlo para lee Además, cuando estamos escribiendo datos en un fichero, el ordenador no lo envía según los vamos introduciendo, pues sería muy lento. El C-64 va acumu lando en cierta parte de su memoria los datos que deseamos grabar, y cuand se llena esta parte de su memoria envía todos los datos a gran velocidad haci la cinta. Si no llegamos a llenar esta parte de su memoria, no envía nada, menos que utilicemos el comando CLOSE, entonces envía el resto de los dato que queden en su memoria hacia la cinta. Así que si desea tener ficheros si errores tendrá que acostumbrarse a utilizar CLOSE.

Con todo esto ya podemos escribir nuestro primer fichero en la Datassette Coloque una cinta en la grabadora, y cuando esté seguro que no hay nada gra bado en ella que sentiría perder, comience con el ejemplo siguiente:

```
10 REM ************************
20 REM * INTRODUCCION DE DATOS *
30 REM ************************
40 OPEN1,1,2"PRUEBA"
50 PRINT"♥"
60 PRINT"TECLEE UN NOMBRE Y PULSE"
70 PRINT"RETURN. SI DESEA FINALIZAR"
80 PRINT"INTRODUZCA EL SIMBOLO #"
90 PRINT:PRINT:PRINT
100 INPUT "NOMBRE";A$
110 IF A$="#" THEN 140
120 PRINT#1,A$
130 GO TO 50
140 CLOSE1
```

Al ejecutar este programa, el Commodore le sacará en pantalla el mensaje PRESS RECORD & PLAY ON TAPE. Pulse las teclas PLAY y RECORD de la grabadora (a la vez) e inmediatamente la pantalla se queda en blanco, mientras que el C-64 empieza a grabar en la cinta la cabecera del fichero, «PRUEBA». Tras unos segundos el ordenador le presenta el mensaje de las líneas 60-80 y ya puede dedicarse a introducir los nombres de sus amigos y pulsar RETURN tras cada uno de ellos. Observe que cuando pulsa RETURN la grabadora no se pone en funcionamiento. Sólo empezará la grabación si ha tecleado ya muchos nombres o si escribe el símbolo # y pulsa RETURN. Con esto el control pasa a la línea 140, que cierra el fichero, y la pantalla volverá a quedar en blanco mientras se realiza la grabación. Realmente es muy lento, pero es que una grabadora es, en sí misma, un dispositivo de muy baja velocidad.

Y ahora, lo crea o no, ya tenemos grabados los nombres de nuestros amigos en la cinta. ¿Cómo podemos comprobarlo? Para ello tenemos que estudiar otra instrucción. Una que nos permita leer los datos grabados.

INPUT# nf, «dato a leer»

nf: Número del fichero del que se van a leer los datos.
«Dato a leer»: Nombre de variable cuyo contenido se va a leer en la cinta.

Compruébelo con el programa siguiente:

```
10 REM *********************
20 REM * LECTURA DE DATOS *
30 REM *********************
40 PRINT"♥"
50 PRINT "PARA RECUPERAR LOS DATOS"
60 PRINT"PULSE CUALQUIER TECLA"
70 GET R$:IF R$="" THEN 70
80 OPEN 1,1,0,"PRUEBA"
```

```
90 PRINT"J"
100 PRINT"LOS NOMBRES GRABADOS SON"
120 INPUT#1,A$
130 PRINT A$
140 IF ST<>64 THEN 120
150 CLOSE 1
```

Antes de ejecutar el programa rebobine la cinta hasta el lugar dondé comenzó la grabación del fichero. De todos modos el programa le da una nueva oportunidad de comprobar si ya tiene preparada la cinta con las líneas 50-70. Quizá sería mejor cambiar este mensaje por algo así como «COMPRUEBE LA CINTA Y PULSE DESPUES CUALQUIER TECLA». Cuando pulse una tecla, aparecerá el mensaje PRESS PLAY ON TAPE. Tras hacerlo, el control pasa a la línea 80, que comienza la búsqueda del fichero (es un proceso bastante lento); en este momento la pantalla se quedará en blanco y cuando vuelva a la normalidad aparecerán en ella los nombres que estaban grabados en la cinta. Ahora ya estamos preparados para crear ficheros con los datos que más nos interesen: nuestra colección de discos, nuestros amigos y sus teléfonos, etc.

En la línea 140 hemos introducido algo nuevo. La línea quiere decir: «si el fichero no se ha terminado vuelve a la línea 120». Pero esto requiere una explicación adicional de ST. Esta es una variable interna del sistema que toma diversos valores, según cuál ha sido el resultado de la última operación de entrada/salida de datos.

Valor de ST	*Significado*
1	Operación correcta
2	Operación correcta.
4	Bloque pequeño. El bloque a leer tiene menos octetos de lo esperado.
8	Bloque grande. El bloque a leer tiene más octetos de lo esperado.
16	Error irrecuperable de lectura.
32	Uno o más bits han sido leídos incorrectamente. Error en la comprobación de la suma.
64	Fin de fichero
128	Fin de la cinta.

Fig. 3.9. Posibles valores de ST.

Otra forma de recuperar los datos de un fichero es mediante la instrucción:

GET#nf, «nombre de variable»

Pruebe a sustituir, en el programa anterior, la línea 120 por:

120 GET#1,A$

Y al volver a realizar la operación de recuperación verá aparecer los nombres de sus amigos, pero una letra debajo de otra. Recuerde que GET sirve para recuperar un solo carácter cada vez. Si quiere verlo en forma correcta, debe poner un símbolo de punto y coma al final de la línea 130:

130 PRINT A$;

Como ejercicio final trate de construir dos programas: uno, para crear un fichero con el nombre y teléfono de sus amigos, y otro, para poder consultarle; compare su solución con la siguiente.

```
3   REM************************************
4   REM*          CARGAR AGENDA           *
5   REM************************************
10 PRINT"JPOSICIONE LA CINTA EN EL
   LUGAR DONDE":PRINT
20 PRINT"DESEE COLOCAR EL FICHERO Y
   PULSE":PRINT
30 PRINT"CUALQUIER TECLA CUANDO LO
   HAYA HECHO":PRINT:PRINT
40 GET A$:IF A$="" THEN 40
50 INPUT"NUMERO MAXIMO DE TELEFONOS";M
60 PRINT"JINTRODUZCA NOMBRE Y
   TELEFONO SEGUN SE":PRINT
70 PRINT"LOS VAYA PIDIENDO. PARA
   TERMINAR PULSE":PRINT
80 PRINT"UNICAMENTE [RETURN] CUANDO
   LE PIDA EL":PRINT
90 PRINT"NOMBRE.":PRINT:PRINT
100 OPEN 1,1,2,"AGENDA"
110 PRINT#1,M
120 N$ = "":INPUT"NOMBRE  ";N$
130 IF N$="" THEN 210
140 INPUT"TELEFONO";T$
150 PRINT
160 PRINT#1,N$
170 PRINT#1,T$
180 M = M-1
190 IF M>0 THEN 120
200 PRINT"ALCANZADO EL MAXIMO DE TELEFONOS"
210 CLOSE 1
```

```
3   REM****************************
4   REM*    CONSULTAR AGENDA      *
5   REM****************************
```

```
10 PRINT"□POSICIONE LA CINTA EN EL
   LUGAR DONDE":PRINT
20 PRINT"COMIENZA EL FICHERO.
   PULSE CUALQUIER":PRINT
30 PRINT"TECLA CUANDO LO HAYA
   HECHO.":PRINT:PRINT
40 GET A$:IF A$="" THEN 40
50 OPEN 1,1,0,"AGENDA"
60 INPUT#1,M
70 DIM N$(M),T$(M)
80 IF ST>3 THEN PRINT "SU AGENDA
   ESTA VACIA.":END
90 I = I+1
100 INPUT#1,N$(I),T$(I)
110 IF ST=0 THEN 80
115 CLOSE 1
120 PRINT"□LA AGENDA YA ESTA
    CARGADA":PRINT
130 PRINT:A$ = "":INPUT"EL TELEFONO
    DE QUIEN",A$
135 IF A$="" THEN END
140 FOR H=1 TO I
150 IF N$(H)=A$ THEN 190
160 NEXT H
170 PRINT"NO TIENE EL TELEFONO DE ";A$
180 GOTO 130
190 PRINT"TELEFONO DE ";A$;": ";T$(H)
200 GOTO 130
```

DIVISION DE TAREAS

En general, cuando atacamos un problema, lo primero que hacemos es desglosar el problema en partes y después empezar a resolver cada una por separado, siempre que sea posible. El lenguaje BASIC nos proporciona dos herramientas para que podamos seguir este esquema de trabajo: las funciones definidas y las subrutinas.

Funciones definidas

Estas nos permiten definir un tipo de actuación sobre una variable, en cierta parte del programa, y después recurrir a esta definición cuando queramos. Para definir una función tenemos que utilizar.

DEF FN «n.f» («n.v») = «expresión»

«n.f»: Es el nombre de la función que vamos a utilizar. Ha de ser un nombre de variable real, pero no puede tener más de cinco caracteres u obtendremos un mensaje de error.
«v.v»: Representa a la variable sobre la que se va a actuar. Ha de ser una variable numérica.
«expresión»: Son las operaciones a efectuar sobre la variable indicada en «n.v».

Para utilizar la función definida con la sentencia anterior tenemos que utilizar el comando:

FN «n.f» («n.v»)

«n.f»: Igual que en DEF FN.
«n.v»: Es la variable con la que se va a efectuar la operación indicada en DEF FN. Su nombre no tiene por qué coincidir con el indicado en DEF FN.

Tras esto habrá advertido que las posibilidades de trabajar con funciones definidas en el Commodore son muy pobres. Sólo se permite un argumento y ha de ser una variable numérica. No obstante, aún podemos utilizarlas para resolver algunos problemas. Suponga que trabajando con números con decimales, deseamos crear una función que se quede con sólo las dos primeras cifras decimales.

```
10 REM ************
20 REM * REDONDEO *
30 REM ************
40 DEF FNC(X)=INT(100*X)/100
50 PRINT"□"
60 PRINT"TECLEE UN NUMERO CON DECIMALES"
   ::INPUT R
70 PRINT:PRINT:PRINT
80 PRINT"REDONDEANDO ESTE NUMERO QUEDA"
   ;FNC(R)
```

La sentencia DEF FN, al igual que DATA, puede estar en cualquier lugar del programa, ya que cuando el ordenador llegue al comando FN localizará el DEF FN para ver qué operaciones ha de realizar con el argumento del FN. Observe además que el nombre del argumento del FN no tiene por qué coincidir con el del DEF FN. Es decir, el argumento del DEF FN es «simbólico» o «representativo». En el ejemplo hemos utilizado las variables R en FN y la X en DEF FN.

Subrutinas

Esta estructura de programación sí que nos permite diseñar nuestros programas en una forma modular que después podamos encajar fácilmente. Además con ella se hace factible la posibilidad de construir módulos de programa (por ejemplo, una rutina de ordenación de listas) que después podemos utilizar en diversos programas. Para estudiar sus características consideremos el siguiente esqueleto de programa:

```
10 REM ********************
20 REM * GESTION DE DATOS *
30 REM ********************
40 PRINT"♥"
50 PRINT TAB(12);"MENU PRINCIPAL"
60 PRINT:PRINT"1.CREACION DE FICHERO"
70 PRINT:PRINT"2.INTRODUCCION DE DATOS"
80 PRINT:PRINT"3.ACTUALIZACION DE DATOS"
90 PRINT:PRINT"4.SACAR LISTADOS"
100 PRINT:PRINT"5.GRABAR FICHERO"
110 PRINT:PRINT"6.CARGAR FICHERO"
120 PRINT:PRINT:PRINT
130 PRINT"PULSE EL NUMERO DESEADO"
140 GET A$:IF A$="" THEN 140
    THEN 140
150 IF ASC(A$)<49 OR ASC(A$)>54
160 A=VAL(A$):PRINT"♥"
170 IF A=1 THEN GOSUB 1000
180 IF A=2 THEN GOSUB 2000
190 IF A=3 THEN GOSUB 3000
200 IF A=4 THEN GOSUB 4000
210 IF A=5 THEN GOSUB 5000
220 IF A=6 THEN GOSUB 6000
230 GO TO 40
1000 REM ******************************
1004 REM * RUTINA PARA CREAR FICHEROS*
1006 REM ******************************
1010 PRINT:PRINT:PRINT
1020 PRINT TAB(10);
     "CREACION DE FICHEROS"
1030 GOSUB 7000
1040 RETURN
2000 REM ******************************
2004 REM *RUTINA PARA INTRODUCIR DATOS*
2006 REM ******************************
2010 PRINT:PRINT:PRINT
2020 PRINT TAB(9);
     "INTRODUCCION DE DATOS"
2030 GOSUB 7000
2040 RETURN
3000 REM ******************************
3004 REM *RUTINA PARA ACTUALIZAR DATOS*
3006 REM ******************************
3010 PRINT:PRINT:PRINT
3020 PRINT TAB(9);
     "ACTUALIZACION DE DATOS"
3030 GOSUB 7000
3040 RETURN
4000 REM ******************************
4004 REM * RUTINA PARA SACAR LISTADOS *
4006 REM ******************************
4010 PRINT:PRINT:PRINT
4020 PRINT TAB(9);
     "OBTENCION DE LISTADOS"
4030 GOSUB 7000
4040 RETURN
5000 REM ****************************
5004 REM * RUTINA PARA GRABAR DATOS *
5006 REM ****************************
5010 PRINT:PRINT:PRINT
5020 PRINT TAB(11);
     "GRABACION DE DATOS"
5030 GOSUB 7000
5040 RETURN
6000 REM ****************************
6004 REM * RUTINA PARA CARGAR DATOS *
6006 REM ****************************
6010 PRINT:PRINT:PRINT
6020 PRINT TAB(9);
     "REPRODUCCION DE DATOS"
6030 GOSUB 7000
6040 RETURN
7000 REM ******************************
7004 REM * RUTINA PARA VOLVER AL MENU *
7006 REM ******************************
7010 PRINT:PRINT:PRINT
7015 FOR I=1 TO 1000:NEXT I
7020 PRINT"DESEA VOLVER AL MENU?(S/N)"
7030 GET R$:IF R$="S" THEN RETURN
7040 IF R$<>"N" THEN 7030
7050 PRINT"♥":PRINT TAB(17);"ADIOS"
7099 END
```

El núcleo principal del programa está en las líneas 10-230. Se presentan siete opciones y usted ha de elegir una pulsando el número correpondiente. El orde-

nador comprueba cuál ha sido el número solicitado y le envía a la subrutina correspondiente:

GOSUB «núm. de línea»

«núm. de línea»: El número de línea en la que comienza la rutina. Si no coincide con algún número de línea del programa obtendrá un mensaje de error.

La instrucción GOSUB viene del inglés GO TO SUBROUTINE (ve a la subrutina), y cuando el ordenador encuentra este comando envía el control de ejecución al número de línea especificado tras el comando.

La estructura de las subrutinas que comienzan en las líneas 1000, 2000, 3000, 4000, 5000 ó 6000 es similar. En realidad se han puesto a modo de ejemplo, indicándole que es aquí donde tendrá que escribir sus propias subrutinas de creación, actualización, grabación, etc. En los ejemplos del programa lo único que hacen es imprimir en pantalla lo que sería el título de la subrutina (línea 1020, 2020,...,6020) y ceder después el control a otra subrutina: la 7000.

Antes de examinar la subrutina 7000 fíjese en el comando RETURN con el que finalizan las subrutinas 1000-6000.

RETURN

Devuelve el control a la línea siguiente a aquella que llamó a la subrutina.

Esta sentencia indica que la subrutina ha finalizado y que el control vuelve a la línea siguiente a la que llamó a la subrutina. Pero fíjese, además, que para realizar esta operación no es necesario indicar el número de línea tras el RETURN, es decir, el ordenador recuerda cuál fue la línea que llamó a la subrutina y devuelve el control a la línea siguiente.

Esto se entiende mejor con la subrutina 7000. Todas las subrutinas anteriores llaman a la 7000, y si nuestra respuesta a la línea 7020 es «S», el RETURN de la línea 7030 devuelve el control a la línea RETURN de la subrutina que realizó la llamada y esta línea devuelve a su vez el control a la línea siguiente a la que le cedió el control. En forma gráfica, la estructura de control del programa anterior la podemos ver en la figura 3.10.

Por ejemplo, si A=1 la línea 170 cede el control a la 1000, después la línea 1030 transfiere a su vez el control a la 7000. Si nuestra respuesta a la línea 7020 es «S», el RETURN de esta línea devuelve el control a la línea 1040, que al ser un RETURN cede a su vez el control a la línea 180, pues fue la línea 170 la que realizó la llamada a la subrutina 1000.

Otro punto a observar en la subrutina 7000 es que el comando RETURN puede estar dentro de una línea multisentencia y además también tras un IF. En general, puede estar situada en cualquier posición correcta de una instrucción normal.

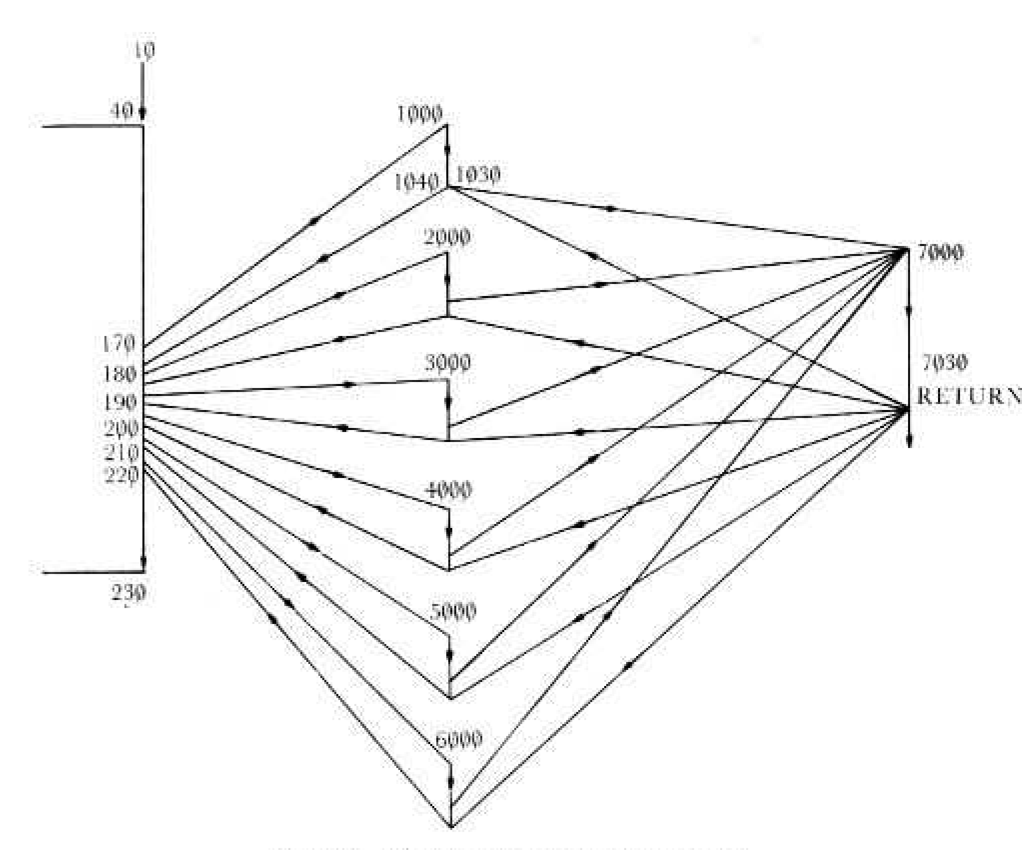


Fig. 3.10. Diagrama de la estructura de control.

La estructura de las líneas 170-220 es tan típica que el C-64 nos proporciona ro comando que puede realizar la función de todas estas líneas a la vez.

ON «expresión» GOSUB «1.er núm. de línea, 2.º núm. de línea,...»

«expresión»: Tal como A o A+4, que el ordenador evalúa y toma la parte entera como resultado. Si el resultado es 1, se cede el control al primer número de línea que aparezca tras el GOSUB. Si el resultado es 2, al segundo número de línea, y así sucesivamente. Si hay seis números de línea tras el GOSUB, entonces «expresión» puede tomar valores del 1 al 6. Un valor de «expresión» fuera de este rango ocasionará que el control pase a la línea siguiente a la que contiene la sentencia ON...GOSUB

Pruebe a introducirlo, borrando las líneas 170, 180, 190, 200, 210 y 220 del rograma anterior y escribiendo en su lugar.

170 ON A GOSUB 1000, 2000, 3000, 4000, 5000, 6000

En realidad, también podríamos borrar la línea 15Ø, pues si el número que pulsamos excede del rango permitido el control pasa a la línea 23Ø. Y además, si pulsamos una letra, la línea 1ØØ introduce un Ø en A.

¿Y por qué utilizar GOSUB en lugar de GO TO? Si utilizamos GO TO para ir a una rutina, después tendremos que finalizar la rutina con un GO TO que devuelva el control a la línea siguiente a la que llamó, y esto impediría el poder utilizar una subrutina desde diversas partes del programa (fíjese en la subrutina 7ØØØ del programa anterior), pues cuando finalice la subrutina, ¿cómo recordaremos a qué lugar del programa tenemos que devolver el control?

Con esto ya hemos dado un repaso a las estructuras más importantes de programación en BASIC con el C-64. Quizá el primer ejercicio que habría que desarrollar es el de rellenar el esqueleto del último programa con los ejemplos que vimos anteriormente sobre creación de ficheros, grabación, etc., con las modificaciones oportunas. ¿Y después? Lo mejor es acudir, ahora que ya está preparado, a algún libro general de ejercicios o a un libro específico sobre el tema de su interés. Es importante no dejar de practicar, pues en muchas ocasiones las cosas se olvidan más fácilmente que se aprenden.

4 Gráficos y sonido

INTRODUCCION

El Commodore 64 ofrece grandes posiblidades en cuanto a la construcción de gráficos y la generación de sonido. Sin embargo, la explicación detallada de todas estas características va más allá de una simple introducción y precisa de un conocimiento más profundo del funcionamiento de nuestro ordenador. No obstante, con el nivel que ya tenemos sí que seremos capaces de crear gráficos y generar sonido a un nivel que dejará asombrados a nuestros amigos y servirá para animar nuestros programas.

GRAFICOS

No existen en el C-64 comandos de BASIC que se encarguen directamente del manejo de los gráficos. Por ello, cuando pasemos del apartado de «Control de la pantalla», tendremos que introducir los comandos PEEK y POKE, que nos permitirán controlar el contenido de la memoria del ordenador, para poder imprimir, colorear y mover los gráficos que deseemos utilizar. Otra opción es no estudiar esta sección de gráficos y adquirir el cartucho de BASIC SIMON. Este es un BASIC muy ampliado con un gran número de instrucciones que han aparecido en otras versiones de BASIC y en particular con palabras clave que nos permiten manejar los gráficos y sonido en una forma mucho más fácil que utilizando las instrucciones PEEK y POKE. El único inconveniente es que ocupa gran parte de la memoria del C-64.

Control de la pantalla

Como ya vimos en el primer capítulo, la pantalla se puede considerar formada por una cuadrícula de 25 filas (de Ø a 24) y 4Ø columnas (de Ø A 39). En cada una de estas posiciones podemos colocar un carácter.

También hemos aprendido a controlar la impresión mediante las funciones TAB y SPC y mediante los separadores «,» y «;». Pero hay un punto que allí mencionamos brevemente y que es importante repasar ahora: la introducción de las acciones de control dentro de los PRINT alfanuméricos.

Por acciones de control nos referimos a las teclas CLR/HOME, INST/DEL, CRSR ↑/↓ y CRSR ←/→. Estas acciones las podemos introducir dentro del PRINT, pero debemos recordar que la forma que tienen de aparecer

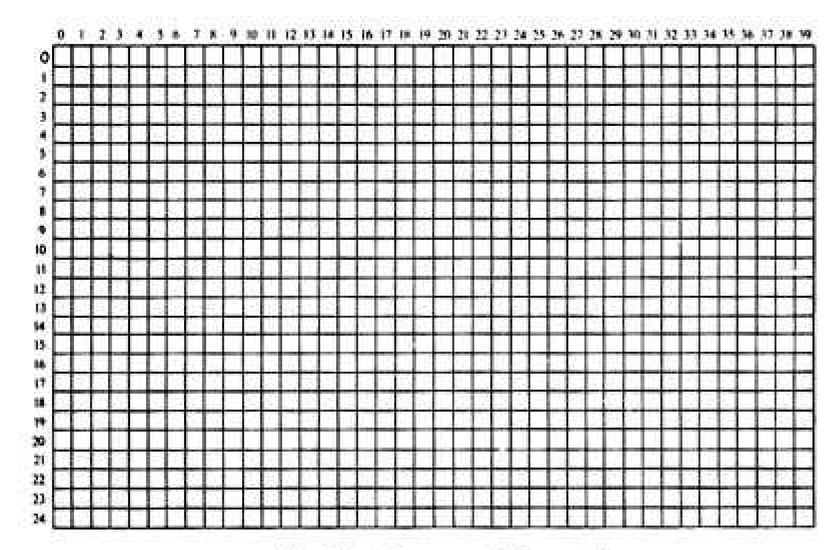


Fig. 4.1. Estructura de la pantalla.

es mediante ciertos símbolos que puede ver en la lista de la figura 4.2. (Para obtenerlos hay que pulsar la tecla, o teclas, correspondientes dentro de las comillas de un PRINT.)

Acción	*Teclas*	*Símbolo*
Cursor a la esquina superior izquierda	CLR/HOME	S
Borrado de pantalla y cursor a la esquina superior izquierda	SHIFT y CLR/HOME	♥
Cursor abajo	CRSR ↑/↓	Q
Cursor arriba	SHIFT y CRSR ↑/↓	●
Cursor a la derecha	CRSR ←/→	]
Cursor a la izquierda	SHIFT y CRSR ←/→	I

Fig. 4.2. Símbolos que representan a ciertas acciones de control.

Si queremos centrar un título en medio de la pantalla podemos utilizar los dos programas siguientes:

```
10 REM *****************
20 REM * PRIMERA FORMA *
30 REM *****************
40 PRINT"♥"
50 PRINT:PRINT:PRINT
60 INPUT"TECLEE LA PALABRA";P$
70 IF LEN(P$)>40 THEN 40
80 PRINT"♥"
90 FOR I=1 TO 11:PRINT:NEXT
100 PRINT TAB(20-LEN(P$)/2);P$
110 GO TO 110
```

```
10 REM *****************
20 REM * SEGUNDA FORMA *
30 REM *****************
40 A$="♥"
50 B$="QQQQQQQQQQQQ"
60 C$="]]]]]]]]]]]]]]]]]]]]"
70 PRINT A$+LEFT$(B$,3)
80 INPUT"TECLEE LA PALABRA";P$
90 IF LEN(P$)>40 THEN 70
100 PRINT A$+LEFT$(B$,12)+
    LEFT$(C$,(20-LEN(P$)/2));P$
110 GO TO 110
```

En principio parece más complicado el segundo programa, sin embargo, según se vaya acostumbrando le resultará mucho más sencillo y observará las grandes posibilidades de gestión de la pantalla que le ofrece el empleo de las acciones de control. En ambos programas medimos la longitud de P$ con objeto de no tratar de centrar en la línea algo cuya longitud es superior a 40 caracteres. En el segundo programa se definen las variables A$, B$, y C$ como unas cadenas de acciones que después fragmentamos mediante LEFT$ para obtener el desplazamiento deseado. El bucle infinito de la línea 110 es, como siempre, para evitar que tras la impresión aparezca el mensaje READY. Para salir de este bucle pulse la tecla RUN/STOP.

Utilizando las acciones de control de impresión podemos escribir primero en la parte inferior de la pantalla, luego en la superior, después en la parte derecha y tras esto en la izquierda. Compruébelo con el programa siguiente:

```
10 REM *********************
20 REM * PUNTOS CARDINALES *
30 REM *********************
40 PRINT"♥";
50 PRINT"QQQQQQQQQQQQQQQQQQQQQQQQ"+
   "]]]]]]]]]]]]]]]]]]]SUR";
60 PRINT"TTTTIIIIIIIIIIIIIIIIIIII"+
   "]]]]NORTE";
70 PRINT"QQQQQQQQQQQ]]]]]]]]]]]]]]]]]]]ESTE";
80 PRINT"]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]"+
   "]]]]]]]]]]]]]]]OESTE"
90 GO TO 90
```

Antes de analizar el programa anterior debe recordar que cuando un PRINT termina sin punto y coma, el cursor que indica en qué posición de la pantalla se va a imprimir la próxima vez salta a la línea siguiente. Para evitar tener que contabilizar este salto colocamos un punto y coma al final de cada PRINT. En la línea 40 borramos la pantalla y colocamos el cursor en la posición 0, 0 (línea 0 columna 0). Para imprimir SUR nos desplazamos 24 líneas hacia abajo y 18 columnas hacia la derecha. Tras esto, para imprimir NORTE, subimos de nuevo a la línea 0 y retrocedemos cuatro columnas, pues NORTE es más largo que SUR y queremos colocarlo centrado. Las palabras ESTE y OESTE las vamos a imprimir en la línea 12. Por ello bajamos 11 líneas y avanzamos 14 columnas, pues NORTE nos dejó en la columna 22, y para imprimir OESTE tenemos que retroceder hasta la columna 0. Fíjese que, aunque el cursor de la pantalla pase, en su desplazamiento, por encima de una palabra, no la borra.

Para un dominio más completo de la pantalla pruebe con el programa siguiente:

```
10 REM ************************
20 REM * CONTROL DE PANTALLA *
30 REM ************************
40 PRINT"⊐"
50 A$="S"
60 B$="QQQQQQQQQQQQQQQQQQQQQQQ"
70 C$="]]]]]]]]]]]]]]]]]]]]"+
   "]]]]]]]]]]]]]]]]]]]]"
80 PRINT A$+B$+"EN QUE LINEA";
90 INPUT L
100 GOSUB 1000
110 PRINT A$+B$+"EN QUE COLUMNA";
120 INPUT C
130 GOSUB 1000
140 PRINT A$+B$+"QUE PALABRA";
150 INPUT P$
160 GOSUB 1000
170 PRINTA$+LEFT$(B$,L)+LEFT$(C$,C)+P$
180 GO TO 80
1000 PRINT A$+B$+"
1010 RETURN
```

Observe cómo utilizamos [S] para devolver el cursor a la posición 0, 0 sin borrar la pantalla. Tras esto podemos situar el cursor en cualquier posición mediante desplazamientos hacia abajo (23 Q invertidas) y hacia la derecha (40 símbolos] invertidos). La subrutina 1000 se utiliza para borrar los mensajes de los INPUT. Sin embargo, este programa no tiene detectores para evitar un manejo indebido, así, si escribe en la línea 12, columna 38, «PEPITO», esta palabra aparecerá partida entre las líneas 12 y 13. Introduzca instrucciones para evitar este tipo de resultado.

Otras acciones de control son las que se obtienen con la tecla INST/DEL, sin embargo, es más difícil diseñar un programa que haga patente cómo pueden utilizarse dentro de los PRINT. Recurramos a un ejemplo ya utilizado:

```
10 REM ****************************
20 REM * IMPRIMIENDO EN PANTALLA *
30 REM ****************************
40 PRINT"⊐"
50 GET A$: IF A$="" THEN 50
60 PRINT A$;
70 GO TO 50
```

Al ejecutar este programa podrá ver cómo aparecen en la pantalla todos los caracteres que teclee. También puede utilizar las teclas de control CLR/HOME, CRSR ←/→ CRSR ↑/↓ e INST/DEL. Sin embargo, como no aparece visible el cursor de impresión, es difícil controlar su movimiento a lo largo del texto que estemos escribiendo. Pruebe, tras escribir unas frases, a utilizar INST/DEL y saque conclusiones sobre el resultado.

Dentro de las comillas del PRINT podemos introducir también las acciones de RVS ON y RVS OFF y los colores. RVS ON y RVS OFF se obtienen manteniendo pulsada la tecla CTRL y pulsando la tecla 9 ó la 0, según la acción deseada. La acción de RVS es que los caracteres afectados aparecerán con el color de la tinta y del papel invertido. RVS OFF restablece la apariencia normal de los caracteres. Dentro de las comillas, RVS ON aparece como una R invertida y RVS OFF como un símbolo de guión invertido.

Acción	*Teclas*	*Símbolo*
RVS ON	[CTRL] y [9]	[R]
RVS OFF	[CTRL] y [0]	[▬]

Fig. 4.3. Símbolos que representan a RVS ON y a RVS OFF.

```
10 REM ****************************
20 REM * INVERSION DE CARACTERES.*
30 REM ****************************
40 PRINT"⊐"
50 PRINT"RINVERTIDO"
55 PRINT:PRINT:PRINT
60 PRINT"NORMAL"
65 PRINT:PRINT:PRINT
70 PRINT"RINVERTIDO  ";
80 PRINT"TODAVIA INVERTIDO"
85 PRINT:PRINT:PRINT
90 PRINT"RINVERTIDO  ▬NORMAL"
100 GO TO 100
```

Observe, líneas 50 y 60, que el efecto de RVS ON sólo afecta al PRINT en el que que está contenido. Cuando se termina de ejecutar la línea 50 se vuelve a pasar a impresión normal. Para evitar esto (ver líneas 70 y 80) hay que terminar el PRINT con un punto y coma, pues esto evita el retorno de carro (RE-

TURN) que tiene lugar en otro caso. En la línea 90 se muestra cómo dentro de una línea PRINT podemos primero pasar a RVS ON y después a RVS OFF, utilizando los símbolos que los representan.

Y después de hacer todo esto dentro de un PRINT, ¡cómo no vamos a poder introducir los colores! Este es el último punto que nos queda por ver para mejorar el aspecto de nuestros mensajes. Como ya vimos en el capítulo 1, los colores los podemos obtener pulsando la tecla CTRL y las teclas numéricas del 1 al 8 ó la tecla **C=** y las teclas del 1 al 8 (ver Fig. 4.4).

Cuando, dentro de unas comillas, pulsamos las teclas para obtener un color aparecerá un símbolo que representa el color deseado. Después, cuando se ejecute el PRINT, su contenido aparecerá en el color indicado.

Color	*Teclas*	*Símbolo*
NEGRO	CTRL y 1	
BLANCO	CTRL y 2	
ROJO OSCURO	CTRL y 3	
AZUL VERDOSO	CTRL y 4	
PURPURA	CTRL y 5	
VERDE OSCURO	CTRL y 6	
AZUL OSCURO	CTRL y 7	
AMARILLO	CTRL y 8	
NARANJA	C= y 1	
MARRON	C= y 2	
ROJO CLARO	C= y 3	
GRIS OSCURO	C= y 4	
GRIS INTERMEDIO	C= y 5	
VERDE CLARO	C= y 6	
AZUL CLARO	C= y 7	
GRIS CLARO	C= y 8	

Fig. 4.4. Símbolos que representan a los colores.

```
10 REM ***********
20 REM * COLORES *
30 REM ***********
40 PRINT"▯"
50 PRINT"■NEGRO"
60 PRINT"▤BLANCO"
70 PRINT"▚ROJO OSCURO"
80 PRINT"◣AZUL VERDOSO"
90 PRINT"▦PURPURA"
100 PRINT"↑VERDE OSCURO"
110 PRINT"←AZUL OSCURO"
120 PRINT"╥AMARILLO"
130 PRINT"▲NARANJA"
140 PRINT"▞MARRON"
150 PRINT"⊠ROJO CLARO"
160 PRINT"▢GRIS OSCURO"
170 PRINT"▩GRIS INTERMEDIO"
180 PRINT"▌VERDE CLARO"
190 PRINT"◆AZUL CLARO"
200 PRINT"⊞GRIS CLARO"
```

Al ejecutar este programa verá que el resultado de las líneas 70, 140 y 150 es ininteligible. Esto se debe a la tonalidad del fondo sobre el que escribimos (AZUL OSCURO), y sólo es necesario cambiar el color del fondo (cosa que aprenderemos muy pronto). La línea 110 escribe en el mismo color (AZUL OSCURO) que el color del fondo, y por ello es imposible ver nada. Hemos escrito en tinta del mismo color que el papel (imagínese escribiendo con tinta negra sobre el papel negro, o con tinta blanca sobre el papel blanco). Observe, finalmente, que tras finalizar la ejecución del programa, el color del cursor queda determinado por el último color que le hemos asignado, en este caso gris claro. Esto no pasaba con RVS ON. El efecto de RVS ON finaliza al terminar un PRINT que no acabe en punto y coma. El efecto de una asignación de color permanece hasta que realicemos otra asignación de color. Compruébelo ejecutando:

```
10 REM *************************
20 REM * EFECTO DE LOS COLORES *
30 REM *************************
40 PRINT"▯"
50 PRINT"⊠ESTO ES ROJO CLARO"
60 PRINT:PRINT"Y ESTO"
70 PRINT:PRINT"Y ESTO."
80 PRINT:PRINT"▌ESTO ES VERDE CLARO"
90 PRINT:PRINT"Y ESTO"
100 PRINT:PRINT"Y ESTO"
```

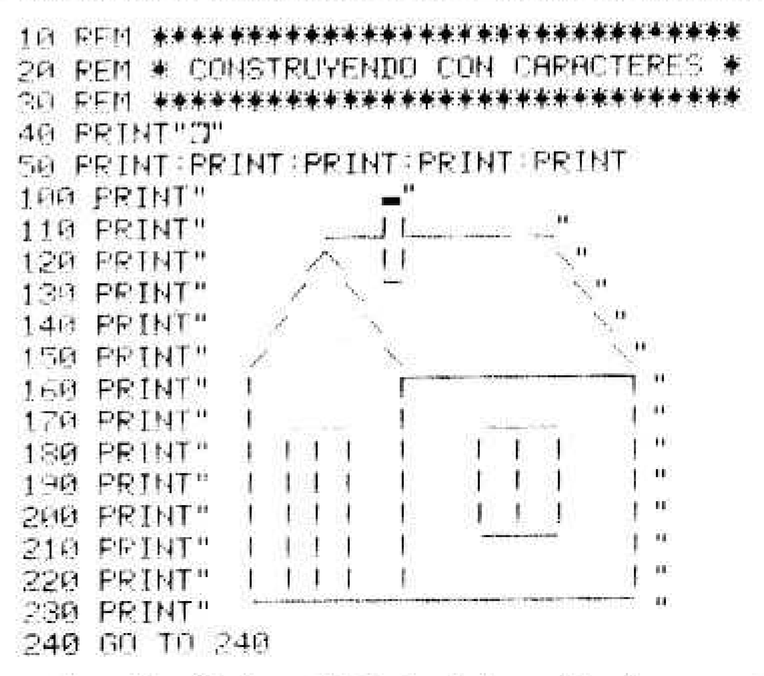
Y fíjese en que, tras la ejecución, el cursor queda de color verde claro. Si desea devolver al cursor su color normal, pulse directamente las teclas **C=** y 7.

Caracteres gráficos

Además de poder teclear letras y números y de introducir la modalidad RVS ON o los colores, vemos que en la falda de las teclas aparecen una serie de ca-

racteres gráficos: rayas de diverso grosor y en diversa situación, los símbolos de la baraja, círculos, cuartos de círculo, etc. Los gráficos situados a la derecha se obtienen pulsando la tecla SHIFT y la tecla deseada, y los situados a la izquierda, utilizando la tecla Commodore (**C=**). (Puede verse una explicación más detallada en la sección sobre el teclado del primer capítulo.)

Estos caracteres gráficos los podemos introducir dentro de las comillas de un PRINT para adornar la presentación de los resultados de cualquiera de nuestros programas o para construir, con varios de ellos, una figura o un dibujo que deseemos presentar. En el programa siguiente se utiliza un conjunto de líneas PRINT para presentar en la pantalla el dibujo de una casa. Como siempre, utilizamos al final un bucle infinito 240 GO TO 240 para evitar que el mensaje de finalización del programa «READY» estropee la estética de la presentación



```
10 REM *******************************
20 REM * CONSTRUYENDO CON CARACTERES *
30 REM *******************************
40 PRINT"♥"
50 PRINT:PRINT:PRINT:PRINT:PRINT
100 PRINT"                    "
110 PRINT"                    "
120 PRINT"                    "
130 PRINT"                    "
140 PRINT"                    "
150 PRINT"                    "
160 PRINT"                    "
170 PRINT"                    "
180 PRINT"                    "
190 PRINT"                    "
200 PRINT"                    "
210 PRINT"                    "
220 PRINT"                    "
230 PRINT"                    "
240 GO TO 240
```

La realización de un dibujo de este tipo es laboriosa, y es aconsejable pensar primero el diseño, lejos del Commodore, y, una vez decidido, volver a la mesa de trabajo para introducirlo mediante el teclado. Así que tome un lápiz, una goma y una cuadrícula de la pantalla, tal como la de la figura 4.1, y comience a trazar el dibujo que desee obtener. Recuerde la diversidad de caracteres gráficos que puede utilizar y tenga en cuenta que, aunque un carácter gráfico ocupe muy poco del espacio de un carácter, por ejemplo, ▐ , sobre esta misma posición del carácter no podrá colocar otro gráfico, por ejemplo ╱ . Esto es, a la hora de realizar su boceto coloque un solo carácter gráfico sobre cada posible posición del cursor.

Con la figura terminada sobre el papel, pase al teclado y comience a introducir el programa. La forma en que se suele hacer es: 1) Mediante las teclas CRSR ↑/↓ y CRSR ←/→ se va desplazando el cursor hasta la posición adecuada y se pulsa el carácter correspondiente. 2) Una vez construida la figura, colocamos el cursor en el margen izquierdo y tecleamos los números de línea y los "PRINT " en cada línea de la figura.

Este método de construir figuras es muy laborioso, pero, una vez realizado, podemos guardar nuestras figuras en diversas subrutinas y luego utilizarlas en programa, por ejemplo, programas educativos, en las que se mezclan figuras con letras y con otras figuras más sencillas construidas mediante métodos que vamos a explicar en la sección siguiente, para obtener el efecto deseado.

Gráficos en movimiento

Como veremos, hay diversas formas de generar movimiento de los gráficos en la pantalla, entre ellas, con el conocimiento que ya tenemos, podemos instrumentar la más rudimentaria.

Considere que deseamos mover una pelota por la pantalla. El carácter gráfico que representa la pelota lo obtenemos pulsando las teclas SHIFT y Q. Así que colocamos la «pelota» en una posición de la pantalla y ahora deseamos moverla. Para hacer el efecto de *movimiento* hay un método muy sencillo. Tras realizar la acción que indique que la pelota ha de moverse; lo primero que hay que hacer es borrar la pelota de su antigua posición, calcular cuál es la nueva posición y después imprimir de nuevo la pelota en este último lugar.

Y ¿qué es eso de realizar la acción que indique que la pelota ha de moverse? En el ejemplo que se expone a continuación se ha resuelto de una forma muy sencilla. Si se pulsa un 1, la pelota se mueve hacia la izquierda, con un 2 se mueve hacia arriba, con un 3 hacia abajo y con el 4 hacia la derecha. Además, la línea 170 se encarga de que, si se pulsa cualquier otra tecla, el ordenador no haga ni caso.

Para calcular la posición en la que hemos de imprimir, lo primero que hacemos es introducir en la variable A$ la acción HOME ([S]), en la variable B$ 24 desplazamientos hacia abajo, CRSR↓([Q]), y en C$ 39 desplazamientos hacia la derecha, CRSR→([ ]]). La primera posición en la que vamos a imprimir la obtenemos al azar en las líneas 110 y 120 (observe que B variará de 0 a 24 y que C variará de 0 a 39), y en la línea 130 introducimos el número de desplazamientos adecuado. La acción HOME ([S]) coloca el cursor en la posición 0, 0 (fila 0 columna 0). Después LEFT$ (B$, B) provoca B desplazamientos hacia abajo y LEFT$ (C$, C) introduce C desplazamientos hacia la derecha. La línea 130 se encarga de imprimir la «pelota» en la posición inicial, y después la línea 170 espera hasta que pulsemos la tecla deseada: 1, 2, 3 o 4. Según el número que pulsemos, la línea 200 enviará el control a una subrutina en la que se calculará la nueva posición de la pelota (antes de esto, en la línea 190 se ha borrado la pelota de su posición anterior). Así, si pulsamos el 1, «izquierda», la subrutina 1000 hace C=C—1, una columna menos, y después devuelve el control a la línea 210, que a su vez lo envía a la línea 130 para imprimir la «pelota» en su nueva posición. Fíjese cómo en la línea 1030 se comprueba también si el valor de la columna, C, ha pasado a ser negativo tras restarle 1. Esto indica que la «pelota» ha salido por el margen izquierdo, y por ello la acción C=39 hace que vuelva a aparecer por el margen derecho.

```
10 REM *************
20 REM * MOVIMIENTO *
30 REM *************
40 PRINT"♥"
50 A$="S"
```

```
60 B$="▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯"
70 C$="▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮"+
   "▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮"
80 REM ********************
90 REM * POSICION INICIAL *
100 REM********************
110 B=INT(RND(1)*25)
120 C=INT(RND(1)*40)
130 PRINT A$+LEFT$(B$,B)+LEFT$(C$,C);"●";
140 REM *************************
150 REM * CONTROL DEL MOVIMIENTO *
160 REM *************************
170 GET M$:IF M$<>"1"AND M$<>"2"AND
    M$<>"3"AND M$<>"4" THEN 170
180 M=VAL(M$)
190 PRINT A$+LEFT$(B$,B)+LEFT$(C$,C);" ";
200 ON M GOSUB 1000,2000,3000,4000
210 GOTO 130
1000 REM *************
1010 REM * IZQUIERDA *
1020 REM *************
1030 C=C-1:IF C<0 THEN C=39
1040 RETURN
2000 REM **********
2010 REM * ARRIBA *
2020 REM **********
2030 B=B-1:IF B<0 THEN B=24
2040 RETURN
3000 REM *********
3010 REM * ABAJO *
3020 REM *********
3030 B=B+1:IF B>24 THEN B=0
3040 RETURN
4000 REM ***********
4010 REM * DERECHA *
4020 REM ***********
4030 C=C+1:IF C>39 THEN C=0
4040 RETURN
```

El punto y coma que aparece tras las líneas 130 y 190 evita el efecto de retorno de carro que tiene lugar tras realizar un PRINT, pues si no, cuando desplazamos el carácter hacia abajo y llegamos a la línea 24 se duplicará la «pelota» por el efecto de desplazamiento hacia arriba de la pantalla. Sin embargo, es más complicado evitar que esto ocurra cuando el carácter llega a la posición 24, 39, pues al imprimir aquí y desplazarse la pantalla obtendremos más de una «pelota». Pruebe a modificar el programa para evitar este efecto.

Manejo de gráficos en la memoria

El contenido que nos presenta el Commodore en la pantalla está guardado dentro de la memoria del ordenador. Así, en las direcciones de memoria 1024-2023 (1000 posiciones = 25 filas × 40 columnas) se almacenan los caracteres que en cada momento nos presenta en la pantalla, y en las direcciones 55296-56295 (1000 posiciones) está el color en que nos presenta cada carácter. Luego, si pudiéramos acceder a estas direcciones de memoria y colocar directamente en ellas el contenido que queramos, esto tendrá que reflejarse en la pantalla. Pero ¿cómo podemos acceder a una dirección de memoria y cambiar su contenido? Tenemos que aprender un nuevo comando: POKE

POKE d, c

d: Es la dirección de memoria a la que queramos acceder. c: Es el valor que deseamos colocar en d.

El valor de c puede variar desde 0 a 255 y se refiere a un número de código de un carácter (ver el apéndice de códigos). Así, POKE 1024, 65 indica que colocamos el carácter (cuyo código es el 65) en la esquina superior izquierda de la pantalla, pues su dirección de memoria es la 1024. Sin embargo, en general, aunque coloquemos un símbolo en la posición 1024, no conseguiremos verlo, a menos que asignemos algún color en la dirección de colores de pantalla correspondiente, en este caso la 55296. Así, el comando directo:

POKE 1024, 65: POKE 55296, 10

hace aparecer un símbolo de PICK, rojo claro, en la esquina superior izquierda de la pantalla. En las figuras 4.5 y 4.6 puede ver las direcciones de memoria que se ocupan de guardar el contenido de cada posición de pantalla y su color, respectivamente.

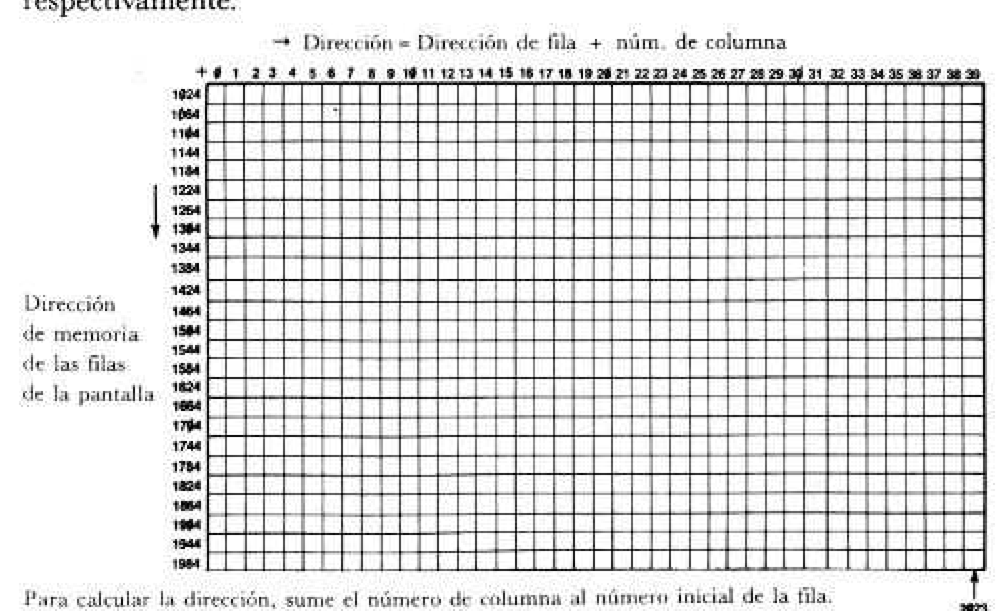
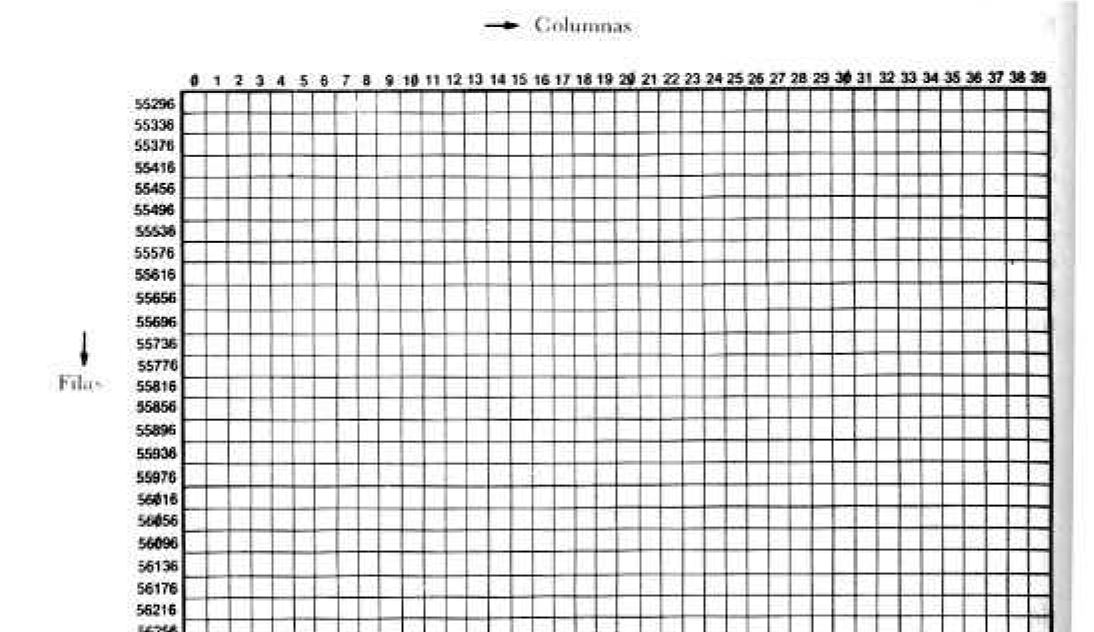


Fig. 4.5. Direcciones de memoria del contenido de las posiciones de pantalla.



Dirección = Dirección de fila + núm. de columna

Fig. 4.6. Direcciones de memoria del color de las posiciones de pantalla.

Aquí nos encontramos con dos problemas:

1) Qué números podemos colocar en la pantalla de caracteres, y
2) qué números podemos colocar en la pantalla de colores.

Para utilizar la pantalla de caracteres hay que observar que los códigos que representan los caracteres en estas direcciones de memoria difieren de los códigos que estamos acostumbrados a utilizar. Estos códigos reciben el nombre de «códigos de pantalla» y se recogen en la segunda parte del apéndice Códigos ASCII.

En cuanto a los colores, el número que podemos colocar en estas direcciones oscila del Ø-15 y su significado se expone en la lista siguiente:

Código	*Color*	*Código*	*Color*
Ø	Negro	8	Naranja
1	Blanco	9	Marrón
2	Rojo	1Ø	Rojo claro
3	Azul verdoso	11	Gris claro
4	Púrpura	12	Gris intermedio
5	Verde	13	Verde claro
6	Azul	14	Azul claro
7	Amarillo	15	Gris oscuro

Fig. 4.7. Códigos para la pantalla de colores.

Así, con el programa siguiente obtendrá una pantalla llena de símbolos y colorido.

```
10 REM ************************
20 REM * PANTALLA PSICODELICA *
30 REM ************************
40 PRINT"J"
50 FOR I=1024 TO 2023
60 POKE I,X
70 X=X+1:IF X=256 THEN X=0
80 POKE I+54272,Y
90 Y=Y+1:IF Y=16 THEN Y=0
100 NEXT
110 GO TO 110
```

La línea 5Ø se encarga de recorrer toda la pantalla de caracteres (de 1Ø24 a 2Ø23) y la línea 6Ø va colocando en cada posición un valor X. De que el rango de X varíe de Ø a 255 se ocupa la línea 7Ø. Paralelamente, en la línea 8Ø vamos asignando un color (Y varía de Ø a 15) a cada posición de la pantalla de colores, lo cual nos permite visualizar los caracteres que va colocando la línea 6Ø. Observe que la distancia entre la primera dirección de la pantalla de caracteres 1Ø24 y la primera dirección de la pantalla de colores 55296 es 54272.

Y para animar un poco más el programa anterior introduzca las líneas.

65 POKE 5328Ø , 2: POKE 53281, 1
85 POKE 5328Ø , 1: POKE 53281, 2

Ahora verá parpadear la pantalla, tanto el borde como el color de fondo, pasando de los colores rojo a blanco, sucesivamente, a una velocidad trepidante. Para estudiar un poco mejor esto es necesario ralentizar la imagen. Introduzcamos las líneas:

75 FOR K=1 TO 5ØØ : NEXT
95 FOR K=1 TO 5ØØ : NEXT

Y veremos cómo alternativamente los colores de fondo y borde van pasando de rojo y blanco a blanco y rojo. ¿Qué es esto de colores de borde y fondo? Fíjese en la figura 4.8.

La imagen que nos presenta el televisor está dividida en dos zonas: el borde y el fondo. Al conectar el C-64, el color del borde es azul claro y el color de fondo es azul oscuro. El fondo es el papel donde escribimos, y, en general, escribimos en tinta de color azul claro sobre el papel de color azul oscuro. Ya hemos visto diversas formas de cambiar el color de la tinta de los caracteres, y ahora, sabiendo cómo variar los colores del papel y del borde, podemos conseguir vistosas presentaciones multicolores.

El color asignado al borde está guardado en la dirección de memoria 5328Ø y el del fondo, en la dirección 53281. Para comprobarlo, pulse con determinación y a la vez las teclas RUN/STOP y RESTORE, para llevar al ordenador a su situación inicial, e introduzca el siguiente comando directo.

POKE 5328Ø , 2: POKE 53281, 5

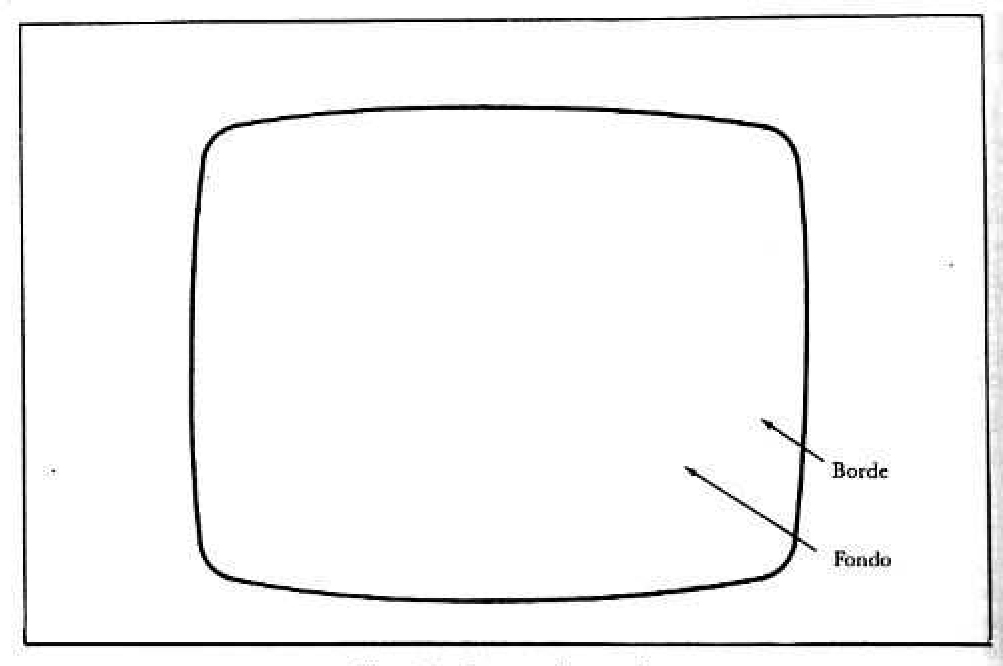


Fig. 4.8. Zonas en la pantalla.

Y además de obtener un borde de color rojo y un fondo de color verde, se encontrará con que no puede leer nada. Tenemos que cambiar el color de la tinta. Si elegimos para la tinta el color negro, basta con pulsar a la vez las teclas CTRL y 1 y el cursor pasará al color negro. Así puede ir probando diversos colores de tinta y de papel hasta encontrar aquella combinación que más le satisfaga. Si en algún momento la mezcla de colores impide que pueda leer lo que está escribiendo, pulse las teclas RUN/STOP y RESTORE, y la presentación volverá a los colores originales.

Los colores que podemos introducir en las direcciones 5328Ø y53281 (borde y fondo) son los mismos que los de la pantalla de colores. Esto es, los números que se pueden colocar van del Ø al 15, y el color correspondiente lo hemos visto antes en la figura 4.7.

Números binarios

Antes de seguir introduciéndonos en este mundo de gráficos y sonido, vamos a hacer una pequeña revisión sobre cómo almacena el ordenador la información. Siempre hemos dicho que en el C-64 podemos colocar números de Ø al 255 en las direcciones de memoria para obtener cierto color o cierto gráfico.

> Cuidado con dedicarse a colocar números indiscriminadamente en cualquier dirección de memoria, pues algunas de éstas las utiliza el ordenador para su funcionamiento, y si cambiamos su contenido podemos encontrarnos con que el ordenador se queda como «en trance», sin responder a ningún comando. Para salir de este estado pulse con decisión las teclas RUN/STOP y RESTORE. El único problema es que puede haber perdido el programa en que estuviera trabajando.

La estructura de estas direcciones de memoria es análoga a un conjunto de ocho celdillas en cada una de las cuales podemos colocar un Ø o un 1. La casilla más a la derecha es la casilla Ø, y la numeración avanza hasta la situada más

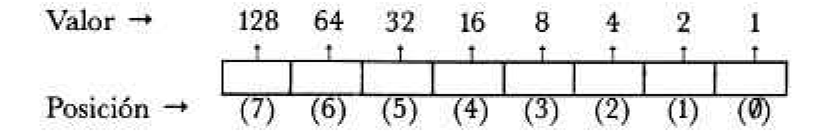


Fig. 4. 9. : Estructura de un byte.

a la izquierda, que es la casilla 7. Como hemos dicho, en cada casilla podemos colocar un Ø o un 1. Digamos que un Ø colocado en una casilla no indica nada. Sin embargo, un 1, colocado en la casilla Ø indica un valor de 1, colocado en la casilla 1 indica un valor de 2, colocado en la casilla 2 indica un 4, colocado en la 3 indica un 8, y así va incrementando (en múltiplos de 2) hasta llegar a un 1 colocado en la casilla 7, que indica un valor de 128 (ver la línea «valor» de la Fig. 4.9).

Casilla	*Relación*	*Valor*
Ø	$2^Ø$	1
1	2^1	2
2	2^2	4
3	2^3	8
4	2^4	16
5	2^5	32
6	2^6	64
7	2^7	128

Fig. 4.1Ø Relación entre el número de casilla y el valor que representa.

El número total almacenado en las ocho casillas es el resultado de sumar el valor de las casillas ocupadas con un 1. Así:

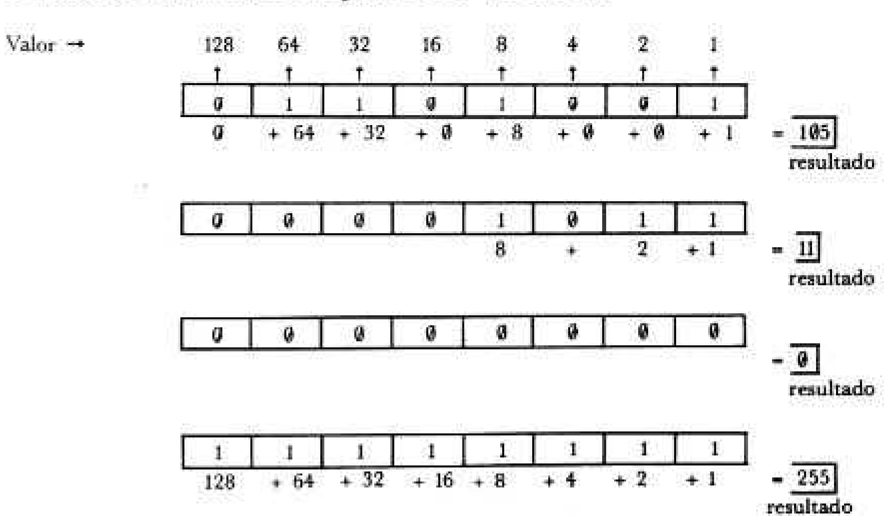


Fig. 4.11. Varios ejemplos de la relación entre números binarios y el resultado decimal.

En la figura 4.11 se exponen algunos ejemplos del número decimal que corresponde a ciertos números binarios. Los dos últimos ejemplos nos indican el número más pequeño, el Ø, y el número más grande, el 255, que podemos representar con un número binario de ocho casillas.

> Cada una de las casillas recibe el nombre de bit (que puede valer Ø ó 1). El conjunto de ocho casillas se llama byte u octeto y puede representar números del Ø al 255. Es decir, con un octeto podemos distinguir 256 números distintos, y esto es lo que se utiliza para almacenar y distinguir los caracteres (ver el apéndice de códigos ASCII). Sin embargo, cuando nos hablan de la memoria de un ordenador nos suelen hablar de K-bytes: por ejemplo, nos dicen: «tiene 16 K». Un K-byte es igual a 1Ø24 bytes. Así si queremos saber cuántos bytes tiene cierto ordenador, haremos, por ejemplo, si son 16 K, el número de bytes es igual a: 16•1Ø24=16384 bytes.

Después de esta breve exposición teórica vamos a entretenernos un poco curioseando en la memoria del C-64. ¿Cómo podemos leer el contenido de una dirección de memoria?

PEEK (d)

> d: Dirección de memoria cuyo contenido queremos leer.

La instrucción PEEK tiene acceso a la dirección de memoria d y lee el número allí almacenado. Pulse las teclas RUN/STOP y RESTORE, para poner el C-64 en su situación original, e introduzca el comando:

PRINT PEEK (5328Ø), PEEK (53281)

Tras pulsar RUN/STOP y RESTORE, los colores de borde y de fondo son, respectivamente, azul claro y azul oscuro. Por ello, al decirle con el comando anterior que lea el contenido de las direcciones 5328Ø y 53281 y las imprima en pantalla esperaríamos ver los números 14 y 6 (fíjese en la Fig. 4.7), y sin embargo el Commodore nos obsequia con los números 254 y 246. ¿Cómo es esto? Los números decimales 254 y 246 se corresponden, respectivamente, con los números binarios.

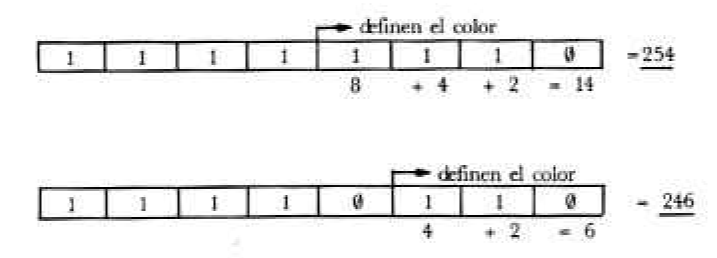


Fig. 4.12. Para definir el color se toman sólo los cuatro dígitos de la derecha.

Fíjese que en ambos casos las cuatro casillas de la izquierda están todas con un 1. Sin embargo, para definir el color el C-64 considera sólo las cuatro casillas de la derecha (1+2+4+8=15) y por ello los números de los colores varían del Ø al 15. Usted puede poner números mayores que 15, mas para calcular el color correspondiente sólo se considerará si los cuatro bits de la derecha contienen ceros o unos.

Antes hemos mencionado que la «pantalla de caracteres» está almacenada en las direcciones 1Ø24-2Ø23 y que la «pantalla de colores» ocupa las posiciones 55296-56295. Pues bien, aunque la «pantalla de colores» siempre permanece en estas direcciones, la posición de la «pantalla de caracteres» podemos calcularla con la fórmula:

PRINT 256*PEEK (648)

Al introducir este comando aparecerá, generalmente, el número 1Ø24. La dirección de memoria 648 es donde se guarda un número que determina en qué lugar de la memoria comienza la memoria de pantalla de caracteres. En este caso

PRINT PEEK (648)

el número almacenado en 648 es el 4, y así 256•4=1Ø24. Esto sirve para almacenar diversas pantallas de caracteres en distintas posiciones de memoria y después, introduciendo un número u otro en la dirección 648, conseguimos que el ordenador nos presente en el monitor (o televisión) una u otra de las pantallas almacenadas en memoria. Así, mientras que estamos observando una pantalla podemos estar diseñando otra dentro del ordenador y visualizarla después.

A continuación le exponemos un programa que visualiza una pelota rebotando dentro de una cancha. Hemos colocado la posición inicial de la pantalla de caracteres, 1Ø24, en la variable CA (línea 6Ø) y la distancia entre la pantalla de caracteres y la pantalla de colores (55296-1Ø24=54272) en la variable DI. El resto del programa intenta explicarse mediante los REM utilizados. En las líneas 26Ø y 27Ø se definen las variables X1 e Y1 como el incremento en la posición horizontal y vertical de la pelota. Si la pelota choca contra los bordes, las líneas 1Ø3Ø y 1Ø4Ø se encargan de cambiar la dirección del movimiento. Para borrar la pelota de la posición anterior, antes de volver a imprimirla, en la línea 34Ø se coloca un espacio en blanco (cuyo código es el 32) en el lugar en que antes se imprimió la pelota. ¡Anímese! y pruebe a poner unas raquetas para poder jugar una partida con los amigos.

```
10 REM **********************
20 REM * JUEGO DE LA PELOTA *
30 REM **********************
40 PRINT"♥"
50 DI=54272
60 CA=1024
80 POKE 53280,13
90 POKE 53281,5
100 REM ************************
110 REM * TRAZADO DEL CONTORNO *
120 REM ************************
```

```
130 FOR I= CA TO CA+39
140 POKE I,102:POKE I+40*24,102
150 POKE I+DI,2:POKE I+DI+40*24,2
160 NEXT I
170 FOR I=CA+40 TO CA+40*23 STEP 40
180 POKE I,102:POKE I+39,102
190 POKE I+DI,2:POKE I+DI+39,2
200 NEXT I
210 REM ******************************
220 REM * POSICION INICIAL ALEATORIA *
230 REM ******************************
240 X=INT(RND(1)*38)+1
250 Y=INT(RND(1)*23)+1
260 X1=1
270 Y1=1
280 POKE CA+DI+40*Y+X,7:
    POKE CA+40*Y+X,81
300 REM **************
310 REM * MOVIMIENTO *
320 REM **************
330 GOSUB1000
340 POKE CA+40*Y+X,32
350 X=X+X1:Y=Y+Y1
360 GO TO 280
1000 REM ***********************
1010 REM * RUTINA DE MOVIMIENTO *
1020 REM ***********************
1030 IF X=1 OR X=38 THEN X1=-X1
1040 IF Y=1 OR Y=23 THEN Y1=-Y1
1050 RETURN
```

GENERACION Y CONTROL DE SPRITES

Además de poder utilizar los caracteres gráficos que se ven en el teclado, el C-64 nos permite diversas formas de crear nuestros propios caracteres gráficos. Esto es, definimos la forma de un gráfico (que bien puede ser un caballo de carreras o un invasor galáctico) y después lo podemos imprimir en la pantalla y moverlo y hacerlo chocar contra otros gráficos, etc. Pero, antes de entusiasmarnos, tenemos que hablar de la *pantalla de alta resolución*. Hasta ahora hemos utilizado la pantalla de *baja resolución* que contenía 40 posiciones horizontales (de 0 a 39) y 25 posiciones verticales (0 a 24). En realidad, cada una de estas posiciones está formada por ocho puntos verticales y ocho puntos horizontales. Por ejemplo, la letra A está formada del modo siguiente:

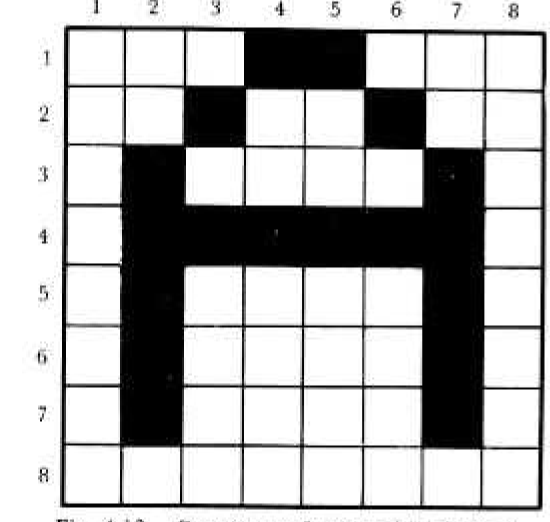


Fig. 4.13. Puntos que forman el carácter A.

Es decir, cada posición de carácter, en baja resolución, está constituida por 8×8=64 puntos. Cuando hablamos de alta resolución nos referimos a cada uno de estos puntos individualmente, y por ello, la pantalla de alta resolución está formada por 320 puntos horizontales (40×8=320) y por 200 puntos verticales (25×8=200).

Definición de SPRITES

Para definir un Sprite, lo primero que tenemos que hacer es coger un lápiz, una goma (o mejor una gran goma) y una cuadrícula como la de la figura 4.14.

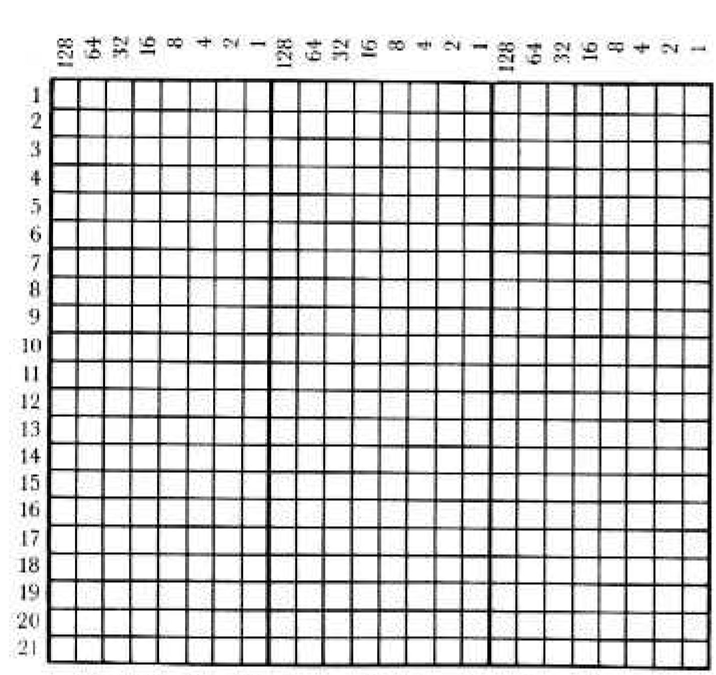


Fig. 4.14. Cuadrícula para el planteamiento de un Sprite.

Como puede ver en la figura, la definición de un Sprite involucra 24 líneas verticales por 21 líneas horizontales. Observe que los 24 puntos de una línea horizontal están agrupados (líneas más gruesas) en tres conjuntos de ocho puntos.

Es decir, para definir cada línea horizontal (24 puntos) tenemos que utilizar tres octetos (8×3=24). Así, la línea 9 de la figura 4.18 tiene la forma:

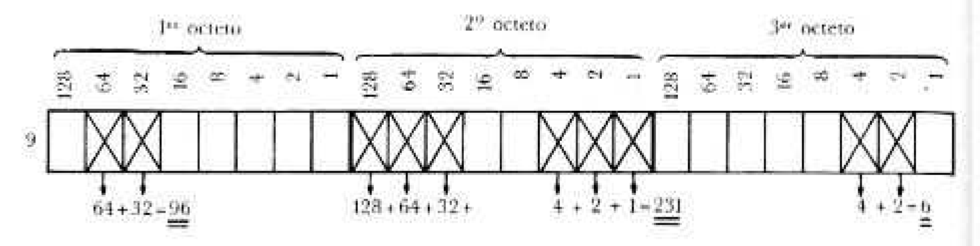


Fig. 4.15. Obtención de los números que representan una fila.

Los puntos ocupados se interpretan como un 1 en esa posición, y los puntos desocupados, como un Ø.

Pues bien, comencemos a dibujar nuestro Sprite. Tomemos la cuadrícula, el lápiz y la goma y a rellenar y borrar puntos hasta que obtengamos un dibujo que nos satisfaga. Después una vez terminado el diseño, consideraremos cada línea horizontal por separado e iremos calculando los tres números que le corresponden (véanse Figs. 4.15 y 4.18). Al final tendremos 63 números (21 filas x 3=63). Y ahora, ¿qué hacemos con estos 63 números?

En el C-64 podemos almacenar hasta tres configuraciones distintas de Sprites. Estas configuraciones se almacenan en las posiciones:

1	832-895
2	896-959
3	96Ø-1Ø24

Fig. 4.16. Direcciones donde se almacenan los Sprites.

Para cada Sprite se dispone de 64 direcciones de memoria, aunque en realidad utilizamos sólo 63.

Pero si bien sólo podemos definir tres formas distintas, podemos utilizar hasta ocho Sprites al mismo tiempo en la pantalla, claro que varios de ellos tendrán que tener la misma forma. Además, cada uno de estos Sprites ha de tener una prioridad. Esto quiere decir que si un Sprite tiene más prioridad que otro, al cruzarse ambos en la pantalla parecerá que el primero pasa por encima del segundo. Las prioridades van del Ø al 7; el Ø tiene la máxima prioridad, y el 7, la mínima. Para indicar la prioridad de un Sprite utilizaremos una dirección distinta para indicar a qué nivel de prioridad nos referimos.

Los números que podemos colocar en estas direcciones (2Ø4Ø-2Ø47) son o bien el 13, el 14 ó el 15. El 13, para referirnos que el Sprite está almacenado en las direcciones 832-895; el 14, para indicar que el Sprite está almacenado en las direcciones 896-959, y el 15, para indicar que está en las posiciones 96Ø-1Ø24.

Dirección	*Prioridad del Sprite*
2Ø4Ø	Ø
2Ø41	1
2Ø42	2
2Ø43	3
2Ø44	4
2Ø45	5
2Ø46	6
2Ø47	7

Fig. 4.17. De la posición utilizada para definir el Sprite depende la prioridad del Sprite.

Vamos a ir desarrollando un ejemplo que nos sirva para asimilar todo esto. El Sprite que queremos dibujar aparece en la figura 4.18.

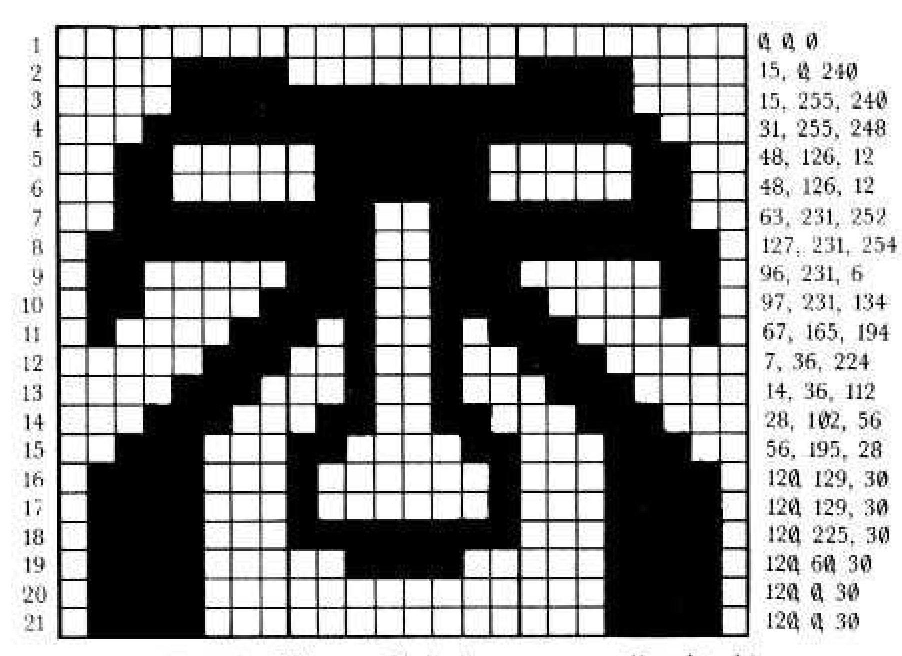


Fig. 4.18. El invasor galáctico (o pantera rosa recién aplastada).

Tras calcular los números que aparecen en el margen derecho de la figura los colocamos ordenadamente en unas líneas DATA, por ejemplo:

```
10000 REM ***********************
10010 REM * FIGURA DEL INVASOR *
10020 REM ***********************
10030 DATA   0,  0,  0
10040 DATA  15,  0,240
```

```
10050 DATA  15,255,240
10060 DATA  31,255,240
10070 DATA  48,126, 12
10080 DATA  48,126, 12
10090.DATA  63,231,252
10100 DATA 127,231,254
10110 DATA  96,231,  6
10120 DATA  97,231,134
10130 DATA  67,165,194
10140 DATA   7, 36,224
10150 DATA  14, 36,112
10160 DATA  28,102, 56
10170 DATA  56,195, 28
10180 DATA 120,129, 30
10190 DATA 120,129, 30
10200 DATA 120,255, 30
10210 DATA 120, 60, 30
10220 DATA 120,  0, 30
10230 DATA 120,  0, 30
```

Después, cargamos estos datos en una de las áreas de memoria dedicada a ello. Elijamos, por ejemplo, la 832-895 y tecleemos la línea:

50 FOR S = 0 TO 62: READ D: POKE 832 + S, D: NEXT S

Según el nivel de prioridad que le queramos dar a nuestro SPRITE, elegimos una de las posiciones 2040-2047 y colocamos en ella el número que indica dónde está almacenado. Recuerde que este número puede ser:

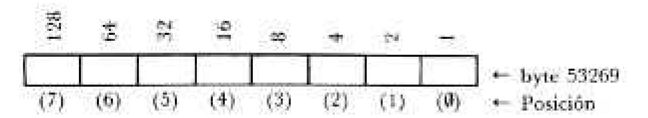
a) 13 si está almacenado en 832-895 (13 × 64 = 832).
b) 14 si está almacenado en 896-959 (14 × 64 = 896).
c) 15 si está almacenado en 960-1023 (15 × 64 = 960).

En nuestro caso elegimos el nivel de prioridad 0 (2040), y como lo hemos guardado en 832-985, tenemos que colocar un 13.

100 POKE 2040, 13

Pues bien, aun con todo esto, cuando ejecutemos el programa, el Sprite no aparecerá. Todavía hemos de hacer dos cosas: decirle al ordenador que active el Sprite de nivel 0 e indicarle en qué posición de la pantalla ha de imprimirlo.

El Commodore tiene una dirección de memoria dedicada a comprobar si los diferentes Sprites están activados o no. Esta dirección es la 53269.



Si queremos activar el Sprite de prioridad 0 tenemos que colocar un 1 en la posición 0.

POKE 53269, 1:

128	64	32	16	8	4	2	1
							1

Si deseamos activar el Sprite de nivel 5 tenemos que colocalr un 1 en la posición (5).

POKE 53269, 32:

128	64	32	16	8	4	2	1
		1					

Si queremos activar a la vez los Sprite de nivel 1, 3 y 5 tenemos que colocar un 1 en cada una de las posiciones (1), (3) y (5)

POKE 53269, 2+8+32:

128	64	32	16	8	4	2	1
		1		1		1	

Para desactivar todos los Sprites podemos introducir POKE 53269, 0 . Así, el efecto de los contenidos del byte 53269 lo podemos resumir en la figura 4.19.

Octeto 53269

El bit	*Controla el Sprite*
(0)	0 (1=Activado)
(1)	1 (1=Activado)
(2)	2 (1=Activado)
(3)	3 (1=Activado)
(4)	4 (1=Activado)
(5)	5 (1=Activado)
(6)	6 (1=Activado)
(7)	7 (1=Activado)

Fig. 4.19. Activación y desactivación de los Sprites mediante el octeto 53269.

En el ejemplo que estamos desarrollando queremos activar el Sprite 0 , y para colocar un 1 en la primera casilla hacemos:

200 POKE 53269, 1

Y ahora lo que nos falta es decirle al ordenador dónde queremos que imprima el Sprite. Para ello tenemos que volver a considerar la pantalla de alta resolución.

Ya hemos mencionado que la pantalla de alta resolución está formada por 320 líneas verticales × 200 líneas horizontales. Sin embargo, cuando nos referimos a un Sprite también hay que contar con los puntos que contiene el borde. Un Sprite impreso en la parte del borde no será visible, pero esto nos permite una entrada suave, progresiva, de los Sprites en la zona de pantalla.

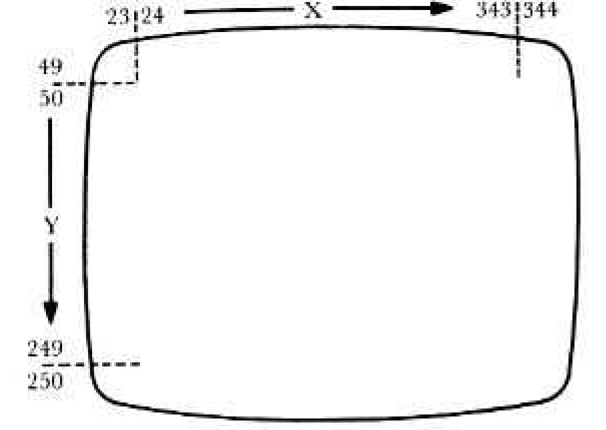


Fig. 4.20. Numeración de la pantalla para los Sprites.

Para cada uno de los ocho Sprites que podemos ver al mismo tiempo, tenemos que dar la coordenada horizontal (X) y la coordenada vertical (Y) en la que queremos que esté situado (observe que las Y aumentan de arriba hacia abajo). Para asignarles coordenadas a los Sprites tenemos que introducirlas en ciertas direcciones de memoria.

Sprite	*Coordenada X*	*Coordenada Y*
0	53248	53249
1	53250	53251
2	53252	53253
3	53254	53255
4	53256	53257
5	53258	53259
6	53260	53261
7	53262	53263

4.21. Posiciones de memoria para colocar las coordenadas X e Y de cada Sprite.

Vamos a probar definitivamente nuestro programa. Asignemos un 100 a las X y un 100 a las Y de nuestro Sprite (recuerde que era el Sprite 0).

300 POKE 53248, 100: POKE 53249, 100

Con lo que el programa total quedaría en la forma:

```
10 REM ************************
20 REM * GENERACION DE SPRITES *
30 REM ************************
40 PRINT"♥"
50 FOR S=0 TO 62:READ D:POKE 832+S,D:NEXT S
100 POKE 2040,13
200 POKE 53269,1
300 FOR X=0 TO 255:POKE 53248,X:
    POKE 53249,100:NEXT
10000 REM **********************
10010 REM * FIGURA DEL INVASOR *
10020 REM **********************
10030 DATA   0,  0,  0
10040 DATA  15,  0,240
10050 DATA  15,255,240
10060 DATA  31,255,240
10070 DATA  48,126, 12
10080 DATA  48,126, 12
10090 DATA  63,231,252
10100 DATA 127,231,254
10110 DATA  96,231,  6
10120 DATA  97,231,134
10130 DATA  67,165,194
10140 DATA   7, 36,224
10150 DATA  14, 36,112
10160 DATA  28,102, 56
10170 DATA  56,175, 28
10180 DATA 120,129, 30
10190 DATA 120,129, 30
10200 DATA 120,255, 30
10210 DATA 120, 60, 30
10220 DATA 120,  0, 30
10230 DATA 120,  0, 30
```

Ejecute el programa para ver qué es lo que aparece en la pantalla. ¡Enhorabuena! Acaba de generar su primer SPRITE, vamos a aprender ahora a divertirnos con él.

Tamaño, color y movimiento

Existen dos formas de crear un Sprite de mayor tamaño. La primera es imprimiendo juntos varios Sprites y la segunda es accediendo a ciertas direcciones de memoria que controlan el tamaño de los Sprites. Estas direcciones son la 53271, para duplicar el tamaño horizontal de los Sprites y la 53277, para duplicar el tamaño vertical. Los números a introducir en estas direcciones dependen del Sprite que deseamos modificar. Así, en general:

Acción	*Afecta al*
POKE d, 1	Sprite 0
POKE d, 2	Sprite 1
POKE d, 4	Sprite 2
POKE d, 8	Sprite 3
POKE d, 16	Sprite 4
POKE d, 32	Sprite 5
POKE d, 64	Sprite 6
POKE d, 128	Sprite 7

Fig. 4.22. Influyendo en el tamaño, color o activación de un Sprite.

Donde d representa la dirección de memoria sobre la que deseamos actuar. Como ya hemos visto, existe una dirección, la 53269, que determina qué Sprites

están activados. Ahora estamos estudiando que las direcciones 53271 y 53277 controlan si se duplica el tamaño, tanto horizontal como vertical, del Sprite que hemos creado (Sprite 0 ; dirección 2040), tendremos que introducir la línea:

330 POKE 53271, 1: POKE 53277, 1

Y nos asustaremos con el aspecto monstruoso de nuestro diseño. Pero ¿qué tal si le damos algo de colorido? Para determinar el color de cada Sprite existe una dirección de memoria para almacenar el color de cada uno de ellos.

Dirección	*Sprite*
53287	0
53288	1
53289	2
53290	3
53291	4
53292	5
53293	6
53294	7

Fig. 4.23. Direcciones para controlar el color de los Sprites.

El color que asignamos a los Sprites se controla mediante el número que almacenamos en cada dirección.

Número	*Color*
0	Negro
1	Blanco
2	Rojo
3	Azul verdoso
4	Púrpura
5	Verde
6	Azul
7	Amarillo
8	Naranja
9	Marrón
10	Rojo claro
11	Gris oscuro
12	Gris intermedio
13	Verde claro
14	Azul claro
15	Gris claro

Fig. 4.24. Números para controlar el color de los Sprites.

Por ello, para darle color rojo claro a nuestro Sprite introduciríamos la línea

340 POKE 53287, 10

pues nuestro Sprite, es el de nivel 0

Para mover el Sprite tenemos que variar las coordenadas X e Y de su posición (ver Fig. 4.20). Si quiere generar un movimiento vertical, borre en primer lugar la línea 300 anterior e introduzca la línea:

360 FOR Y= 0 TO 255: POKE 53248, 160 : POKE 53249, Y: NEXT Y

Ejecute el programa, verá aparecer el Sprite por el borde superior y desaparecer por el borde interior. Divertido, ¿verdad? Sin embargo, el control del movimiento horizontal no es tan sencillo. Podemos, en principio, borrar la línea 360 anterior e introducir la línea:

360 FOR X= 0 TO 225: POKE 53248, X: POKE 53249, 100 : NEXT X

Y al ejecutarlo veremos aparecer el Sprite por el margen derecho y avanzar, pero se para antes de desaparecer por el margen izquierdo ¿Por qué? Muy sencillo, ya hemos visto que el número de puntos horizontales es de 320, pero en la dirección 53248, que controla el movimiento horizontal del SPRITE 0 , sólo podemos colocar números del 0 al 225 y por ello se detiene antes de llegar al final de la pantalla. Esto es fácil de arreglar. Cuando queramos que la coordenada X de uno de nuestros Sprites llegue a valer más de 225, tenemos que indicárselo al ordenador en la dirección de memoria 53264. En esta posición colocaremos uno de los números. 1, 2, 4, 8, 16, 32, 64 ó 128, según queramos que la coordenada X de los Sprites 0, 1, 2, 3, 4, 5, 6 ó 7, respectivamente, sea mayor que 225.

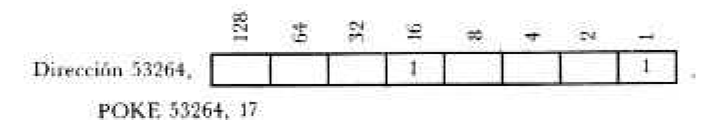


Fig. 4.25. Esta acción indica que la posición horizontal de los Sprites 0 y 4 es igual a la coordenada X que pongamos en el byte de dirección (53248 y 53256) más 255.

En nuestro caso queremos que, después de que el Sprite 0 recorra las posiciones horizontales de 0 a 225, siga su dirección horizontal hasta desaparecer por el lado derecho. Para ello, tras la línea 360 anterior, teclee:

370 POKE 53264, 1: FOR X= 0 TO 86: POKE 53248, X: NEXT X

Y verá cómo el Sprite desaparece por el margen derecho. Si desea volver a ejecutar el programa, más vale que introduzca el comando:

POKE 53264, 0

Pues si el valor de la dirección 53264 sigue siendo 1, al volver a ejecutar el programa, el Sprite comenzará a aparecer a partir de la coordenada X=255.

El programa desarrollado hasta ahora se presenta a continuación:

```
 10 REM ***************************
 20 REM * GENERACION DE SPRITES *
 30 REM ***************************
 40 PRINT"♥"
 50 FOR S=0 TO 62:READ D:POKE 832+S,D:
    NEXT S
100 POKE 2040,13
200 POKE 53269,1
300 REM *******************************
310 REM * TAMANNO COLOR Y MOVIMIENTO *
320 REM *******************************
330 POKE 53271,1:POKE 53277,1
340 POKE 53287,10
```

```
360 FOR X=0 TO 255:POKE 53248,X:
    POKE 53249,100:NEXT X
370 POKE 53264,1:FOR X=0 TO 86:
    POKE 53248,X:NEXT X
400 GO TO 400
10000 REM *********************
10010 REM * FIGURA DEL INVASOR *
10020 REM *********************
10030 DATA   0,  0,  0
10040 DATA  15,  0,240
10050 DATA  15,255,240
10060 DATA  31,255,240
10070 DATA  48,126, 12
10080 DATA  48,126, 12
10090 DATA  63,231,252
10100 DATA 127,231,254
10110 DATA  96,231,  6
10120 DATA  97,231,134
10130 DATA  67,165,194
10140 DATA   7, 36,224
10150 DATA  14, 36,112
10160 DATA  28,102, 56
10170 DATA  56,195, 28
10180 DATA 120,129, 30
10190 DATA 120,129, 30
10200 DATA 120,255, 30
10210 DATA 120, 60, 30
10220 DATA 120,  0, 30
10230 DATA 120,  0, 30
```

Sprites en multicolor y colisiones

Y para animar algo más nuestro juego, el C-64 nos da la posibilidad de presentar nuestros Sprites en diversos colores. Para indicar qué Sprite va a estar en diversos colores tendremos que utilizar la dirección de memoria 53276:

Dirección: 53276

bit=1	*Sprite afectado*
Ø	Ø
1	1
2	2
3	3
4	4
5	5
6	6
7	7

Fig. 4.26. Bits que indican si un Sprite es multicolor o no.

Para asignar la mezcla de coloridos que utilizan los Sprites se utilizan las direcciones 53284 y 53285. Introduzca en el programa anterior, por ejemplo, las líneas:

```
110 POKE 53276, 1
120 POKE 53284, 1
340 POKE 53287, C
380 C=C+1
390 POKE 53264, 0: GO TO 340
400 GO TO 340
```

Y compruebe la diversidad de colores que pueden ofrecerle los Sprites. Sustituya la línea 12Ø anterior por:

```
120 POKE 53285, 1
```

Y amplíe la posibilidad de colorido de los Sprites.

Para detectar las colisiones entre los Sprites se utiliza la dirección de memoria 53278. Esta dirección de memoria está normalmente a Ø, pero si colisionan dos Sprites se coloca en ella la suma de los números que definen los niveles de los Sprites. Así, si colisionan el Sprite 3, al que corresponde un 8, y el Sprite 6, al que corresponde un 64, el número almacenado en 53278 será el 72 (64+8=72).

En general los Sprites tienen prioridad sobre el texto que aparece en la pantalla, y si hemos dejado algo escrito, veremos cómo el Sprite pasa por encima de estos caracteres. Esta prioridad, o no, de los Sprite se almacena en la dirección 53275. Según si queremos quitar la prioridad al Sprite Ø ó al 1 ... ó al 7, pondremos un 1 en el bit Ø, o en el 1, ... , o en el 7 del byte 53275.

Además podemos detectar si un Sprite ha entrado en contacto con un carácter de la pantalla. Para ello se utiliza el octeto de la posición 53279. Si el Sprite Ø colisiona con un carácter, se activa el bit Ø de la dirección 53279.

Esperamos que con esto sea capaz de diseñar sus propios juegos con los Sprites. En todo caso recuerde que la única forma de aprender es practicando, equivocándose y rectificando. Apunte en su libreta cada cosa nueva que descubra, y sea ordenado. De poco le vale apuntar si luego no se acuerda dónde.

Para que le sirva de ayuda le exponemos a continuación un resumen de las direcciones de memoria que hemos utilizado:

Sprite	*Definición*	*Posición X*	*Posición Y*	*Color*
Ø	2040	53248	53249	53287
1	2041	53250	53251	53288
2	2042	53252	53253	53289
3	2043	53254	53255	53290
4	2044	53256	53257	53291
5	2045	53258	53259	53292
6	2046	53260	53261	53293
7	2047	53262	53263	53294

Un 1 en el bit	*Dirección*	*Indica*
i	53264	La coordenada X del Sprite i es superior a 255.
i	53269	Activado el Sprite i
i	53271	Doble tamaño horizontal del SPRITE i.
i	53275	Prioridad sobre el texto del SPRITE i.
i	53276	El SPRITE i es multicolor.
i	53277	Doble tamaño vertical del SPRITE i.
i y j	53278	Los SPRITES i y j han colisionado.
i	53279	Colisión del Sprite i con un carácter.

SONIDO

La síntesis de sonidos es una de las características más sobresalientes del C-64. Las posibilidades de éste son equiparables a las que proporcionan sintetizadores profesionales. El corazón de la producción de sonido en el C-64 es un chip llamado SID, del inglés *Sound Interface Device,* que traduciríamos como dispositivo generador de sonido, capaz de llevar a cabo por sí solo la producción del sonido que le ha encargado el computador, dejando a éste libre para la realización de otras tareas.

Sin embargo, como en el caso de los gráficos, el BASIC del C-64 no posee sentencias específicas para el manejo del sonido y es necesario colocar, mediante POKE, varios valores en las direcciones adecuadas del SID para obtener el efecto apetecido.

Un poco sobre música

Las características sonoras del C-64 fueron concebidas principalmente para la producción de música. A lo largo de este capítulo, normalmente nos referimos a los sonidos, entendiendo como tales las notas musicales, a pesar de que las posibilidades del C-64 no se limitan a ellas, como ponen de manifiesto los efectos sonoros que acompañan a muchos de los programas de juegos existentes en el mercado. Sin embargo, los principios de producción de cualquier tipo de sonidos son idénticos y no se pierde generalidad concentrándose en las notas musicales.

Antes de comenzar a estudiar en detalle la generación de sonidos en el C-64 vamos a repasar algunos conceptos que serán importantes para comprenderla.

Lo que permite distinguir dos notas diferentes, emitidas por un mismo instrumento, son sus frecuencias, es decir, el número de oscilaciones por segundo de la onda sonora, lo que en términos musicales se corresponde con la altura o tono de la nota. Por ejemplo, la nota LA de la octava central de piano corresponde a un sonido de una frecuencia de 44Ø oscilaciones/segundo, mientras que la nota DO de la misma octava corresponde a una frecuencia de aproximadamente 262 oscilaciones/segundo. Nuestro oído es capaz de reconocer que la frecuencia de la nota DO es menor que la frecuencia de la nota LA y que, por tanto, DO es más grave que LA. A la unidad de medida de frecuencia suele llamarse también ciclo/segundo o hertzio, que se abrevia como Hz.

El oído humano puede percibir sonido de frecuencias entre 2Ø y 2ØØØ Hz, pero la mayoría de las notas producidas por instrumentos convencionales poseen alturas que entran dentro del rango de ocho octavas que maneja el C-64 y que abarca frecuencias entre Ø y 3996 Hz.

El timbre es la propiedad del sonido que hace que dos notas de la misma altura emitidas por instrumentos diferentes suenen de forma distinta. Por ejemplo, si se escucha tocar el LA central con un piano y después con una flauta, no será posible decir si el primer sonido es más o menos agudo que el segundo, ya que ambos tienen la misma altura (naturalmente sólo en el caso de que los dos instrumentos estén afinados), pero aun así podrá decirse que primero sonó un piano y luego una flauta.

El timbre se relaciona con el contenido de armónicos que posee el sonido, es decir, con la cantidad e intensidad de las otras oscilaciones que acompañan a la oscilación fundamental. La oscilación fundamental posee la frecuencia correspondiente al tono del sonido, y los armónicos tienen frecuencias múltiplos de ella. Si la frecuencia fundamental es f, sus armónicos tendrán frecuencias 2xf, 3xf, 4xf, etc; por ejemplo, en el caso del LA central que antes se mencionó los armónicos tendrán frecuencias de 88Ø, 132Ø, 176Ø Hz, etc. ¡Pero un momento! Si los armónicos correspondientes a un tono dado son siempre los mismos, ¿cómo es que caracterizan el timbre del sonido? El timbre no está determinado sólo por la presencia o no de armónicos en el sonido, y de hecho dos instrumentos que produzcan los mismos armónicos pueden sonar muy diferentes. Lo que realmente define al timbre es la intensidad de cada uno de los armónicos en relación con la intensidad del tono fundamental, y esto sí que es característico de cada instrumento.

Un sonido con un tono y un timbre dados puede ser emitido con más o menos intensidad. La intensidad o volumen del sonido depende de la amplitud de las oscilaciones que lo constituyen: un sonido es tanto más intenso cuanto más amplias son las vibraciones que lo originan. Esta propiedad, al contrario que el tono y el timbre, no es estática, sino que cambia a lo largo del tiempo, como se verá más adelante. También depende de la distancia de la fuente del sonido, disminuyendo al aumentar la separación, debido al amortiguamiento que el aire ejerce sobre las ondas sonoras.

La última característica a tener en cuenta es que el sonido se mantiene un cierto tiempo después que nace y antes de desaparecer por completo. Este tiempo en que se oye se conoce como su duración.

Y bien, sin más dilación pasemos al estudio de las capacidades sonoras del Commodore-64.

El SID

Las posiciones de memoria desde 54272 a 543ØØ corresponden a los 29 registros del SID. Cada uno de estos registros tiene una función específica que se verá a lo largo de este apartado. De momento, nos concentraremos en la estructura de este circuito, aspecto básico para su comprensión.

El SID dispone de tres voces independientes, esto es, puede producir simultáneamente hasta tres sonidos distintos. Cada una de las voces se controla mediante siete registros, todas ellas de forma muy similar. De los ocho registros restantes, cuatro de ellos tienen funciones que afectan a la vez a las tres voces, y los otros cuatro, funciones especiales. Sigamos ahora la producción de sonidos dentro del SID y veamos en qué puntos podemos influir.

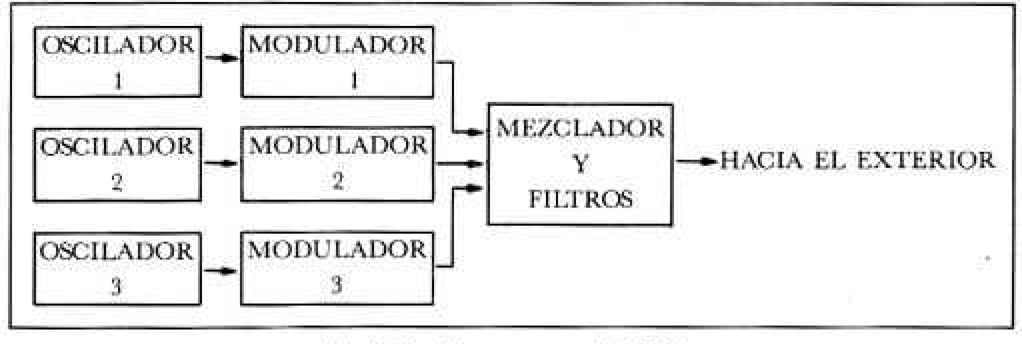


Fig. 4.27. Esquema general del SID.

El sonido nace como una oscilación eléctrica en una parte del SID que se conoce como oscilador. El SID tiene tres osciladores y de ahí que posea tres voces. La frecuencia y la forma de esta oscilación determinan, respectivamente, la altura y el timbre del sonido final.

El SID dispone de cinco registros por cada voz para controlar estas características de la oscilación original, dos dedicados a la frecuencia y tres a la forma.

Ahora bien, un sonido normalmente no aparece con una intensidad determinada y se mantiene así hasta extinguirse. Por el contrario, según un esquema mucho más general, la intensidad de una nota, emitida por un instrumento musical, va cambiando mientras suena. Durante su primer período, que se denomina de ataque o subida, el volumen del sonido crece desde el nivel de no audición, correspondiente al momento en que se produce la nota, hasta el volumen máximo, que alcanzará en toda su duración. A continuación, el sonido empieza a descender de intensidad hasta el nivel de mantenimiento, que alcanza en el tiempo llamado de caída. Viene luego un tercer período denominado de sostenimiento o mantenimiento, es decir, el tiempo en que permanece al volumen del mismo nombre, y finalmente cae de nuevo, hasta extinguirse, durante el período de relajación. A la curva de la figura 4.28, que describe la evolución de la intensidad del sonido en el tiempo, se le llama envolvente.

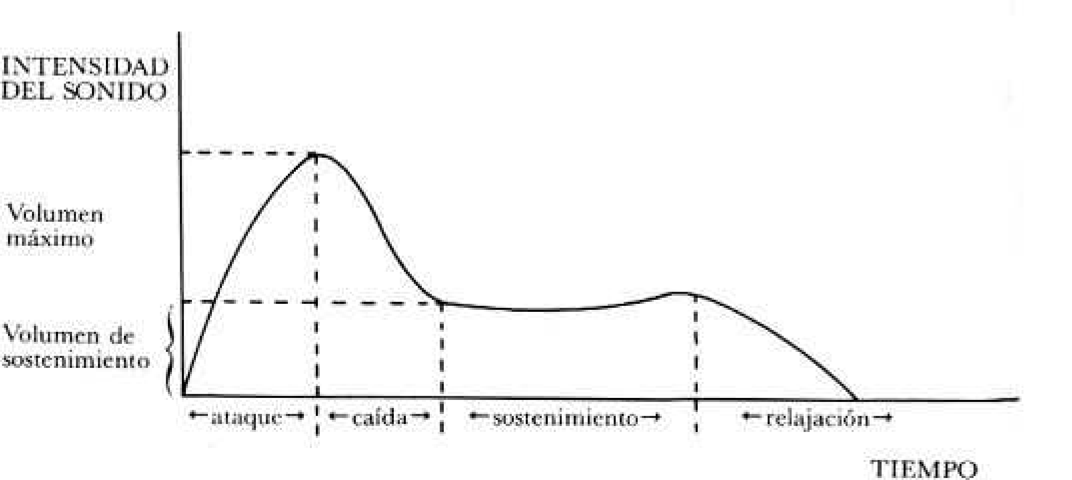


Fig. 4.28.

En la construcción del SID se ha tenido en cuenta esta característica del sonido musical y se le ha provisto de un circuito que la reproduce automáticamente, el modulador de amplitud. El SID tiene también tres moduladores de amplitud, uno por cada voz, siendo controlado cada uno de ellos por dos nuevos registros. La señal eléctrica nacida en un oscilador se hace pasar a continuación por el correspondiente modulador de amplitud, del que sale «modulada» de acuerdo con los parámetros dados a éste.

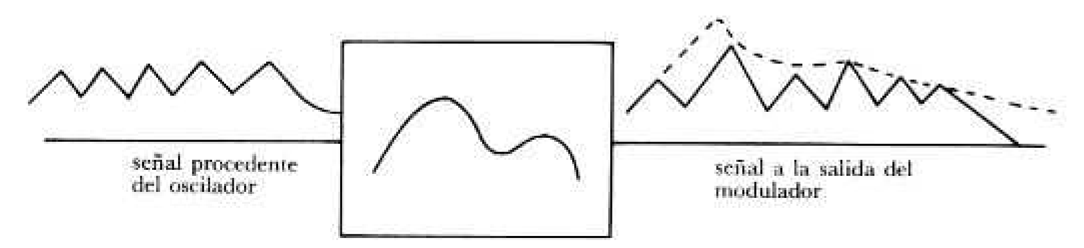


Fig. 4.29. Cómo afecta la modulación en amplitud a una oscilación de forma triangular.

Las salidas de los moduladores pueden filtrarse para eliminar frecuencias no deseadas. El C-64 posee tres filtros distintos que pueden activarse, o no, por separado. También se puede elegir separadamente si la señal de cada oscilador va a ser o no filtrada, pero si se opta por filtrarla lo será a la vez por todos los filtros activos. La frecuencia de corte define el punto en que centra el filtrado que, dependiendo del tipo de filtro, podrá activar de cada una de las siguientes formas:

— Se dejan pasar sin modificación todas las oscilaciones que tengan frecuencia superior a la frecuencia de corte, pero se filtran las frecuencias menores, es decir, se reduce la intensidad (atenúa) de aquellas oscilaciones que tengan una frecuencia más pequeña que la frecuencia de corte, de ahí que a este tipo de filtro se le llame pasa-altos.
— Al contrario, se reduce la intensidad de las frecuencias mayores que la de corte, pero no se modifican las oscilaciones de frecuencia inferior. Es el filtro pasa-bajos.
— En un efecto combinado de los dos anteriores se atenúan todas las frecuencias, salvo las que se encuentran en un margen más o menos estrecho alrededor de la frecuencia de corte, de ahí que a ese tipo de filtro se le llame pasa-banda.

El SID tiene un filtro de cada uno de estos tipos que pueden actuar por separado, o bien por parejas, o los tres a la vez, según los que se encuentren activos. Además puede controlarse la intensidad del filtrado, esto es, si la atenuación es mayor o menor sobre las frecuencias afectadas. A esto se le llama control de resonancia: cuanto mayor sea la resonancia mayor será la atenuación por parte de los filtros.

Finalmente, todas las señales se mezclan y se hacen pasar por un amplificador que controlará el volumen global del sonido; cuatro registros del SID permiten el control de los filtros y el volumen.

¡Bueno, no espere más! Introduzca ya el siguiente programa y ejecútelo. El C-64 le mostrará alguna de sus capacidades para la producción de sonidos interpretando para usted una famosa canción de cuna. Preocúpese ahora sólo de oír al ordenador y vuelva luego al texto para leer la explicación del funcionamiento del programa.

Si se ha dado mucha prisa en introducir el programa y no ha leído antes este párrafo es posible que no haya oído nada. El C-64 envía sus sonidos al altavoz del televisor, por lo que deberá ajustar previamente el mando del volumen del mismo al nivel al que desee escucharlos.

```
10 REM**********************************
20 REM*         CANCION DE CUNA        *
30 REM**********************************
40 SID = 54272
50 FOR I = SID TO SID+24
60 POKE I,0
70 NEXT I
80 READ T
90 READ NN
100 DIM A(NN),B(NN)
110 FOR I = 1 TO NN
120 READ A(I),B(I)
130 NEXT I
140 POKE SID+24,15
150 POKE SID+6,240
160 POKE SID+4,17
170 READ D
180 IF D<0 THEN 240
190 READ N
200 POKE SID,B(N)
210 POKE SID+1,A(N)
220 FOR I = 1 TO D*T: NEXT I
230 GOTO 170
240 POKE SID+24,0
250 END
500 DATA 160
600 DATA 8
610 DATA 22,227,25,177,28,214,30,141
620 DATA 34,75,38,126,43,52,45,198
700 DATA 1,3,1,3
710 DATA 4,5,1,3,1,3
720 DATA 4,5,1,3,1,5
730 DATA 2,8,2,7,2,6
740 DATA 2,6,2,5,1,2,1,3
750 DATA 4,4,1,2,1,3
760 DATA 4,4,1,2,1,4
770 DATA 1,7,1,6,2,5,2,7
780 DATA 4,8,1,1,1,1
790 DATA 2,8,2,6,2,4
800 DATA 4,5,1,3,1,1
810 DATA 2,4,2,5,2,6
820 DATA 2,3,2,5,1,1,1,1
830 DATA 2,8,2,6,2,4
840 DATA 4,5,1,3,1,1
850 DATA 0.67,4,0.67,5,0.67,4,2,3,2,2
860 DATA 4,1
999 DATA -1
```

Antes de entrar en la descripción del uso de los registros del SID, estudie brevemente, y a modo de introducción, cómo trabaja este programa, ya que le servirá de base para poder apreciar los distintos efectos que puede introducir.

En la variable SID de la línea 40 introducimos el valor correspondiente a la primera posición de memoria asignada a registros del SID a fin de hacer más cómoda la referencia a los mismos.

El bucle que empieza en la línea 50 y termina en la 70 se encarga de poner a 0 todos los registros cuyas direcciones nos interesan, de la 54272 a la 54296, con ello conseguimos limpiar el chip de sonido.

La sentencia READ T de la línea 80 lee el primer DATA, que es el situado en la línea 500 , es decir, se introduce en la variable T el valor 160 , que es la duración de una corchea.

En la línea 90 , READ NN lee el segundo DATA, situado en la línea 600 y por tanto la variable NN toma el valor 8, que es el número de notas diferentes que constituyen la canción.

En la línea 100 dimensionamos dos listas; A (8) y B (8), que utilizaremos para que contengan las partes alta y baja de las frecuencias (se aclarará este punto más adelante).

Con el bucle situado desde la línea 110 hasta la 130 leemos los 16 siguientes números contenidos en sentencias DATA (líneas 610 y 620), asignando a las variables A (1), A (2), ... , A (8) los valores correspondientes a la parte alta de la frecuencia de las notas y a B (1), B (2), ... , B (8) los correspondientes a la parte baja de la frecuencia, por ejemplo, A (1) contendrá el valor 22 y B (1) el 227, que son respectivamente los valores correspondientes a las partes alta y baja de la nota FA de la octava cuarta (consúltese Fig. 4.30).

La dirección SID+24 (línea 140) controla el volumen general que ponemos al máximo con el valor 15.

El valor 240 que introducimos en la dirección de memoria SID+6 (línea 150) ajusta los valores elegidos de sostenimiento y relajación para la voz 1. Espere un poco y los entenderá.

La dirección SID+4 (línea 160) controla la forma de onda de la voz 1, única utilizada en este programa, y el 17 es el correspondiente a la forma triangular. Pronto se le explicarán las formas posibles.

En las líneas 170 a la 230 se «leen y hacen sonar» todas las notas. La condición «IF D<0» de la línea 180 se pone para que si el último valor leído es el —1, el control pase a la línea 240, en la que se pone a 0 el volumen, con lo cual se consigue que deje de sonar.

Las líneas 200 y 210 sitúan en las direcciones 54272 (SID) y 54273 (SID+1), respectivamente, la parte baja y alta de la frecuencia de cada nota. Y el bucle de la línea 220 es un bucle de espera, que será distinto para cada nota y proporcional a su duración.

En la línea 250, END indica el final del programa; detrás no hay nada ejecutable.

El resto de las líneas de la 500 a la 999 son DATA.

No borre este programa de la memoria de su ordenador. Más tarde introducirá modificaciones en él para que pueda observar el efecto de cada registro del chip de sonido.

Nota	*Octava*	*Alta frecuencia*	*Baja frecuencia*
DO	0	1	18
DO sostenido	0	1	35
RE	0	1	52
RE sostenido	0	1	70
MI sostenido	0	1	90
FA sostenido	0	1	110
FA sostenido	0	1	132
SOL	0	1	155
SOL sostenido	0	1	179
LA	0	1	205
LA sostenido	0	1	133
SI	0	2	6
DO	1	2	37
DO sostenido	1	2	69
RE	1	2	104
RE sostenido	1	2	140
MI	1	2	179
FA sostenido	1	2	220
FA	1	2	8
SOL	1	3	54
SOL sostenido	1	3	103
LA	1	3	155
LA sostenido	1	2	210
SI	1	4	12
DO	2	4	73
DO sostenido	2	4	139
RE	2	4	208
RE sostenido	2	5	25
MI	2	5	103
FA	2	5	185
FA sostenido	2	6	16
SOL	2	6	108
SOL sostenido	2	6	206
LA	2	7	53
LA sostenido	2	7	163
SI	2	8	23
DO	3	8	147
DO sostenido	3	9	21
RE	3	9	159
RE sostenido	3	10	60
MI	3	10	205
FA	3	11	114

Nota	*Octava*	*Alta frecuencia*	*Baja frecuencia*
FA sostenido	3	12	32
SOL	3	12	216
SOL sostenido	3	13	156
LA	3	14	107
LA sostenido	2	15	70
SI	3	16	47
DO	4	17	37
DO sostenido	4	18	42
RE	4	19	63
RE sostenido	4	20	100
MI	4	21	154
FA	4	22	227
FA sostenido	4	24	63
SOL	4	25	177
SOL sostenido	4	27	56
LA	4	28	214
LA sostenido	4	30	141
SI	4	32	94
DO	5	34	75
DO sostenido	5	36	85
RE	5	38	126
RE sostenido	5	40	200
MI	5	43	52
FA	5	45	198
FA sostenido	5	48	127
SOL	5	51	97
SOL sostenido	5	54	111
LA	5	57	172
LA sostenido	5	61	126
SI	5	64	188
DO	6	68	149
DO sostenido	6	72	169
RE	6	76	252
RE sostenido	6	81	161
MI	6	86	105
FA	6	91	140
FA sostenido	6	96	254
SOL	6	102	194
SOL sostenido	6	108	223
LA	6	115	88
LA sostenido	6	122	52
SI	6	129	120

Nota	*Octava*	*Alta frecuencia*	*Baja frecuencia*
DO	7	137	43
DO sostenido	7	145	83
RE	7	153	247
RE sostenido	7	163	31
MI	7	172	21Ø
FA	7	183	25
FA sostenido	7	193	252
SOL	7	2Ø5	133
SOL sostenido	7	217	189
LA	7	23Ø	176
LA sostenido	7	244	1Ø3
SI	7	253	46

Fig. 4.30. Las notas y sus frecuencias.

Control del SID

Después de estudiar el programa anterior es posible que le haya parecido muy complicado hacer música con el C-64. Lo cierto es que no es fácil, pero si se sigue un método es posible hacerlo. En este apartado se le darán unas reglas que le permitirán construir sus propios programas musicales sin que le surjan dificultades insalvables..

Es bueno, antes de realizar ninguna otra operación, «limpiar» los registros del sonido, como hacen las líneas 5Ø y 7Ø del programa anterior, de lo contrario pueden aparecer sonidos inesperados. Después de esto tendrá que poner el volumen al nivel que desee utilizar, ya que, de realizar esta operación tras las otras, el ordenador puede sorprendentemente quedar callado. Para definir el volumen introduzca un número entre Ø y 15 en la dirección 54296, o bien SID+24 si hace SID=54272. Introduzca las siguientes líneas en su programa.

```
135 FOR VOL=15 TO 1 STEP —2
245 NEXT VOL
```

Y modifique la línea 14Ø para que sea ahora:

```
14Ø POKE SID+24, VOL
```

Ejecute nuevamente el programa y compruebe cómo va disminuyendo la intensidad del sonido a cada interpretación de la canción.

Tras el volumen debieran establecerse los parámetros del modulador de amplitud para cada una de las voces que se vayan a usar. Para hacerlo, seleccione el nivel de sostenimiento a utilizar, que deberá estar entre Ø y 15, y de la tabla, los valores de ataque, caída y relajación que correspondan a los tiempos correspondientes que se requieran. Coloque ahora estos valores en los registros de ataque-caída y sostenimiento-relajación de las voces elegidas que corresponden, respectivamente, a las direcciones 54277 y 54278 (SID+5 y SID +6) para la voz 1, 54284 y 54285 (SID+12 y SID+13) para la voz 2 y 54291 y 54292 (SID+19 y SID+2Ø) para la voz 3. Puede hacer esto con las sentencias:

$$\text{POKE SID} + \begin{Bmatrix} 5 \\ 12 \\ 19 \end{Bmatrix}, \ A*16+C$$

$$\text{POKE SID} + \begin{Bmatrix} 6 \\ 13 \\ 2Ø \end{Bmatrix}, \ S*16+R$$

Valores de ataque, caída y relajación

Valor	*Tiempo de ataque*	*Tiempo de caída y relajación*
Ø	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	2Ø4 ms
7	8Ø ms	24Ø ms
8	1ØØ ms	3ØØ ms
9	25Ø ms	75Ø ms
10	5ØØ ms	1.5 s
11	8ØØ ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

Fig. 4.31.

Siendo A el valor para el ataque, C el de caída, S el de sostenimiento y R el de relajación. Se le brinda más adelante un programa con el que podrá experimentar, modificando la envolvente del sonido que se produce.

A continuación, y aunque no se haya hecho así en el programa anterior, es el momento de fijar la frecuencia del sonido que se desea producir. Para ello es necesario, como podrá observar en las líneas 2ØØ y 21Ø del programa, colocar dos valores a sendos registros del SID. En la figura 4.30 puede observar que a cada nota corresponden dos números, uno que se ha llamado parte baja de la frecuencia y otro al que denominamos parte alta. Pues bien, para determinar la frecuencia debe ponerse el valor de la parte baja de la frecuencia en la dirección 54272 (SID+Ø) si lo es para la voz 1, en la 54279 (SID+7) si lo es para la 2 ó en la 54286 (SID+14) si se trata de la voz 3. Asimismo, la parte alta de la frecuencia deberá ir, respectivamente, a una de las direcciones 54273 (SID+1), 5428Ø (SID+8) ó 54287 (SID+15).

Para hacer sonar la nota sólo es necesario ya definir la forma de onda que habrá de producir el oscilador seleccionado. En el C-64 hay cuatro posibles formas de onda a cada una de las cuales corresponde un valor a colocar en el registro 54276 (SID+4) para la primera voz, en el 54283 (SID+11) para la segunda o en el 5429Ø (SID+18) para la tercera voz.

Los nombres y valores de estas formas de onda se recogen en la figura 4.32.

Formas de onda

Nombre	*Comienzo de la nota*	*Fin de la nota*
Triangular	17	16
Diente de sierra	33	32
Cuadrada	65	64
Ruido	129	128

FIG. 4.32.

Otra vez le proponemos probar el efecto de las distintas formas de onda, para lo cual tendrá que hacer los siguientes cambios en el programa:

— Introduzca las líneas:

```
135 FOR F=4 TO 7
245 NEXT F
```

— Devuelva la línea 14Ø a su forma original:

```
14Ø POKE SID+24, 15
```

— Y modifique la línea 16Ø para que quede:

```
16Ø POKE SID+4, 2 F+1
```

Ahora ejecute el programa y, ¡fíjese en los diferentes timbres!

En la figura 4.32 verá que para cada forma de onda se recogen dos números, uno en la columna «comienzo de la nota» y otro, igual al anterior menos uno en todos los casos, que figura como «fin de la nota». La razón es que cuando se introduce el primero de estos números, pongamos el 17, en el registro de forma de onda correspondiente se inicia el ciclo de la envolvente: comienza el periodo de ataque que terminará un cierto tiempo después de acuerdo al valor definido según la figura 4.31, después vendrá la caída, que durará, igualmente, el tiempo correspondiente de dicha figura, y a éste le seguirá el periodo de sostenimiento en el nivel elegido (entre Ø y 15) que durará... ¿Cuánto durará? Pues justo hasta que el registro de forma de onda pase a tener (porque se ponga en él con POKE) el valor de «fin de la nota». En ese momento comenzará el periodo de relajación, cuya duración dependerá del valor elegido de la figura 4-31.

Ahora bien, en el caso de la onda rectangular aún hay más. En efecto, este tipo de onda es, realmente, un conjunto de tipos de onda. En la figura 4.33 puede ver el aspecto de una oscilación de este tipo: periódicamente pasa de un estado en el que es máxima (pulso) a un estado en el que es mínima.

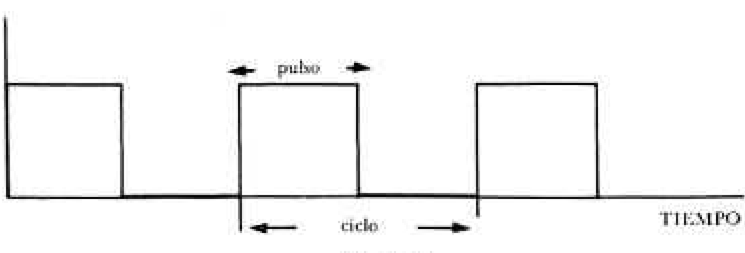


FIG. 4.33.

El SID permite controlar el ancho del pulso, para lo que dispone de dos nuevos registros por voz: el de la parte baja del ancho de pulso (registros 54274 ó SID+2, 54281 ó SID+3 y 54288 ó SID+16), en el que pueden introducirse valores entre Ø y 255, y el de la parte alta del pulso (registros 54275 ó SID+3, 54282 ó SID+1Ø y 54289 ó SID+17), cuyos valores han de estar entre Ø y 15.

Pruebe el efecto de cambiar el ancho del pulso introduciendo las siguientes modificaciones en su programa:

— Introduzca las líneas:

```
135 FOR P=Ø TO 15
165 POKE SID+3, P
245 NEXT P
```

— Modifique la línea 16Ø para que aparezca como:

```
16Ø POKE SID+4, 65
```

Ejecute ahora el programa y escuche cómo va cambiando el timbre.

Aún hay otro medio de modificar el timbre del sonido generado, y es mediante el empleo de los filtros. Para programar los filtros comience por seleccionar las voces a filtrar y el nivel de resonancia que desea (debe estar entre Ø y 15). Multiplique el valor de la resonancia por 16 y súmele 1 para filtrar la voz 1, 2 para filtrar la voz 2 y/o 4 para filtrar la voz 3. Coloque el resultado en el registro 54295 (SID+23), al nivel que desee para el volumen, súmele ahora 16 si quiere activar el filtro pasa-bajos, 32 si quiere activar el filtro pasa-bajos, 32 si quiere activar el filtro pasa-banda y/o 64 si desea activar el filtro pasa-altos, y coloque el valor resultante en el registro de volumen, esto es, en el registro 54296 (SID+24).

Para terminar, deberá definir la frecuencia de corte. Para ello tiene que introducir un valor entre Ø y 255 en el registro 54294 (SID+32) y un valor de Ø en el 54293 (SID+21). El primer valor será la parte alta de la frecuencia de corte, y el segundo, su parte baja. Para obtener el timbre adecuado tendrá que experimentar varios valores para la frecuencia de corte.

En la figura 4.34 se resumen las funciones de cada registro del SID y se le da una breve «receta» para que le sirva de guía al emplearlos.

Quedan por comentar las funciones de los registros 54297 a 54300 (SID+25 a SID+28). Estos registros pueden ser leídos con PEEK, pero no escritos con POKE. Los registros 54297 y 54298 (SID+25 y SID+26) se utilizan para el control de mandos de juegos y no tienen aplicación en sonido.

El registro 54299 (SID+27) contiene en todo momento la salida del oscilador 3 y permite producir brillantes efectos sonoros, aunque para ello es preciso, en general, programar en código de máquina.

Finalmente, el registro 543ØØ (SID+28) contiene en cada instante el valor de la envolvente de la tercera voz y se le puede aplicar lo indicado para el registro anterior.

Dirección			*SID+*			*Función*	*Valor en el registro*
1	2	3	1	2	3		
54272	54279	54286	0	7	14	Parte baja de la frecuencia	0-255 (tabla)
54273	54280	54287	1	8	15	Parte alta de la frecuencia	0-255 (tabla)
54274	54281	54288	2	9	16	Parte baja de ancho de pulso	0-255
54275	54282	54289	3	10	17	Parte alta de ancho de pulso	0-15
54276	54283	54290	4	11	18	Forma de onda	Forma + 1 si se pone / 0 si se quita
54277	54284	54291	5	12	19	Ataque-caída	16•Ataque+caída
54278	54285	54292	6	13	20	Sostenimiento-relajación	16•sostenimiento+relajación
	54293			21		Parte baja de la frecuenc. de corte	0-7
	54294			22		Parte alta de la frecuenc. de corte	0−255
	54295			23		Resonancia y filtros	Reson. (0-15•16+ •1 si filtra voz 1 •2 si filtra voz 2 •4 si filtra voz 4
	54296			24		Tipo de filtro y volumen	64 Si HP 32 Si BP+VOL. (0-15) 16 Si LP (0-15)
	54297			25		Potenciómetro 1	Leídos con PEEK
	54298			26		Potenciómetro 2	
	54299			27		Oscilador 3	
	54300			28		Envolvente 3	

Fig. 4.34

Algunos programas sonoros

Como colofón, le presentamos dos programas que le permitirán profundizar en las características del sonido del C-64.

Empezaremos por un programa para experimentar con los efectos del modulador de amplitud.

```
100 REM*********************************
110 REM*DEMOSTRACION DEL EFECTO DEL EG*
120 REM*********************************
122 PRINT"▯ATAQUE","CAIDA","SOSTEN",
    "RELAX"
124 FOR N = 1 TO 41:B$= B$+" ":NEXT N
130 REM DATOS DE LAS NOTAS
132 DATA 97,8,104,9,143,10,48,11,143,12,
    24,14,210,15
134 DATA 210,15,24,14,143,12,48,11,143,
    10,104,9,97,8
150 REM VOLUMEN MAXIMO
155 POKE 54272,0 : POKE 54273,0
160 POKE 54296,15
170 REM VALORES INICIALES DE ADSR
180 A% = 6:D% = 6:S%= 6:R% = 6
190 PRINT "■■■";B$;"▯";A%,D%,S%,R%
196 RESTORE
198 FOR N = 1 TO 14
200 REM COLOCAR PARAMETROS DEL EG
210 POKE 54277,A%*16+D%
220 POKE 54278,S%*16+R%
230 REM COMENZAR EL SONIDO
232 READ L%,H%
234 POKE 54272,L% : POKE 54273,H%
240 POKE 54276,33
250 M = 100:GOSUB 800
290 REM TERMINAR EL SONIDO
300 POKE 54276,32
302 M = 20:GOSUB 800
310 REM GENERAR NUEVO SONIDO
312 NEXT N
320 GOTO 196
500 REM RUTINA DE CAMBIO
510 IF A$>="■" THEN ON ASC(A$)-132 GOSUB
    540,550,560,570,580,590,600,610
520 PRINT "■■■";B$;"▯";A%,D%,S%,R%
530 RETURN
540 A% = A%-(A%<15) : RETURN
550 D% = D%-(D%<15) : RETURN
560 S% = S%-(S%<15) : RETURN
570 R% = R%-(R%<15) : RETURN
580 A% = A%+(A%>0) : RETURN
590 D% = D%+(D%>0) : RETURN
600 S% = S%+(S%>0) : RETURN
610 R% = R%+(R%>0) : RETURN
800 REM ESPERAR UN POCO
810 FOR T = 1 TO M
820 GET A$ : IF A$<>""THEN GOSUB 500
830 NEXT T
840 RETURN
```

Seguramente se esté preguntando si ninguno va emplear más de una voz. Pues sí, éste. Aquí tiene una melodía para Navidad.

```
10 REM*********************************
20 REM*          NOCHE DE PAZ        *
30 REM*********************************
```

```
40 SID = 54272
50 FOR I = SID TO SID+24
60 POKE I,0
70 NEXT I
80 READ T
90 READ NN
100 DIM A(NN),B(NN)
110 FOR I = 0 TO NN
120 READ A(I),B(I)
130 NEXT I
131 READ NN
132 DIM AA(NN),BB(NN)
133 FOR I = 0 TO NN
134 READ AA(I),BB(I)
135 NEXT I
140 POKE SID+24,15
150 POKE SID+6,240
160 POKE SID+13,64
165 POKE SID+11,17
167 IF D>0 THEN 211
170 READ D
180 IF D<0 THEN 240
190 READ N
192 POKE SID+4,17
200 POKE SID,B(N)
210 POKE SID+1,A(N)
211 IF DD>0 THEN 220
212 READ DD
213 IF DD<0 THEN 240
214 READ N
215 POKE SID+7,BB(N)
216 POKE SID+8,AA(N)
217 POKE SID+11,17
220 FOR I = 1 TO T*0.9: NEXT I
221 IF D=1 THEN POKE SID+4,16
222 IF DD=1 THEN POKE SID+11,16
223 FOR I = 1 TO T*0.1: NEXT I
225 D = D-1: DD = DD-1
230 GOTO 167
240 POKE SID+24,0
250 END
500 DATA 150
600 DATA 10
610 DATA 21,154,22,227,25,177,28,214,32,94
620 DATA 34,75,38,126,43,52,45,198,19,63,17,37
630 DATA 7
640 DATA 14,24,8,97,10,143,12,143
650 DATA 11,48,6,71,7,233,9,104
700 DATA 3,2,2,1,2,2,1,3,2,2,2,3
710 DATA 6,0,2,1,2,2,2,3
720 DATA 3,2,2,1,2,2,1,3,2,2,2,3
730 DATA 6,0,2,1,2,2,2,3
740 DATA 4,6,2,5,2,6,2,6,2,7
750 DATA 6,4,2,5,2,6,2,7
760 DATA 4,5,2,1,2,2,2,5,2,3
770 DATA 6,2,2,1,2,2,2,3
780 DATA 4,3,2,1,2,4,2,3,2,0
790 DATA 3,5,2,1,2,4,1,4,2,3,2,0
800 DATA 3,2,2,1,2,2,1,3,2,2,2,3
810 DATA 6,0,2,1,2,2,2,3
820 DATA 4,3,2,1,2,4,2,3,2,0
830 DATA 3,5,2,1,2,4,1,4,2,3,2,0
840 DATA 3,2,2,1,2,2,1,3,2,2,2,3
850 DATA 6,0,2,1,2,2,2,3
860 DATA 4,6,2,5,2,6,2,6,2,7
870 DATA 3,8,2,5,2,6,1,6,2,4,2,7
880 DATA 6,5,2,1,2,2,2,3
890 DATA 6,7,2,1,2,2,2,3
900 DATA 3,5,2,1,2,2,1,2,2,0,2,3
910 DATA 3,2,2,1,2,2,1,1,2,9,2,3
920 DATA 12,10,12,3
999 DATA -1
```

Apéndice A: Otros periféricos

¿Ha llegado hasta este apéndice? ¡Enhorabuena! Usted ha decidido superar la etapa dedicada a los juegos y pequeños programas de entretenimiento y convertir su Commodore en una potente máquina de cálculo y gestión. En primer lugar, vamos a estudiar el dispositivo de discos; ese artefacto que da una nueva potencia y velocidad a nuestro trabajo. Después, realizaremos una revisión de la utilización de la impresora para aquellos que hayan decidido que desean aprovechar también la posibilidad de presentar sus trabajos mediante el C-64 y de sacar listados de sus programas que les permitan una revisión de éstos sin tener que perder la vista ante la pantalla.

LA UNIDAD DE DISCOS

Instalación

Al desempaquetar la unidad de discos tendrá que encontrar los siguientes elementos:

1. La unidad de discos.
2. Un cable gris para conectar la unidad de discos al enchufe de la pared.
3. Un cable negro para conectar la unidad de discos al C-64.

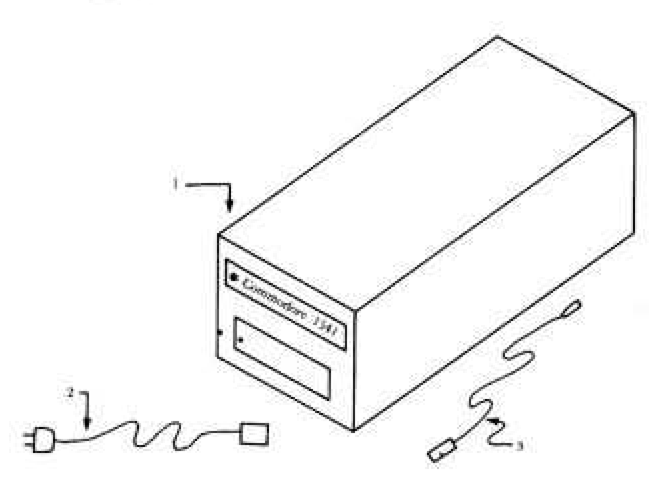


Fig. A.1. Equipo básico del dispositivo de discos.

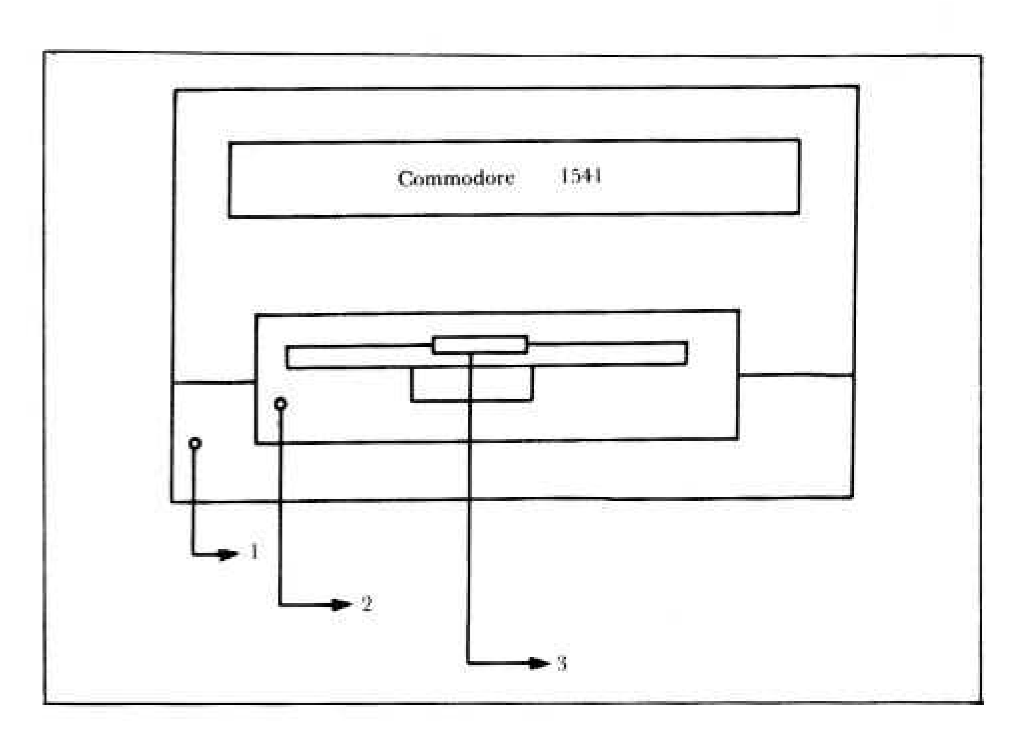


Fig. A.2. Vista frontal. 1. Luz conexión a fuerza (verde). 2. Luz de funcionamiento del disco (roja). 3. Boca de entrada del disquete.

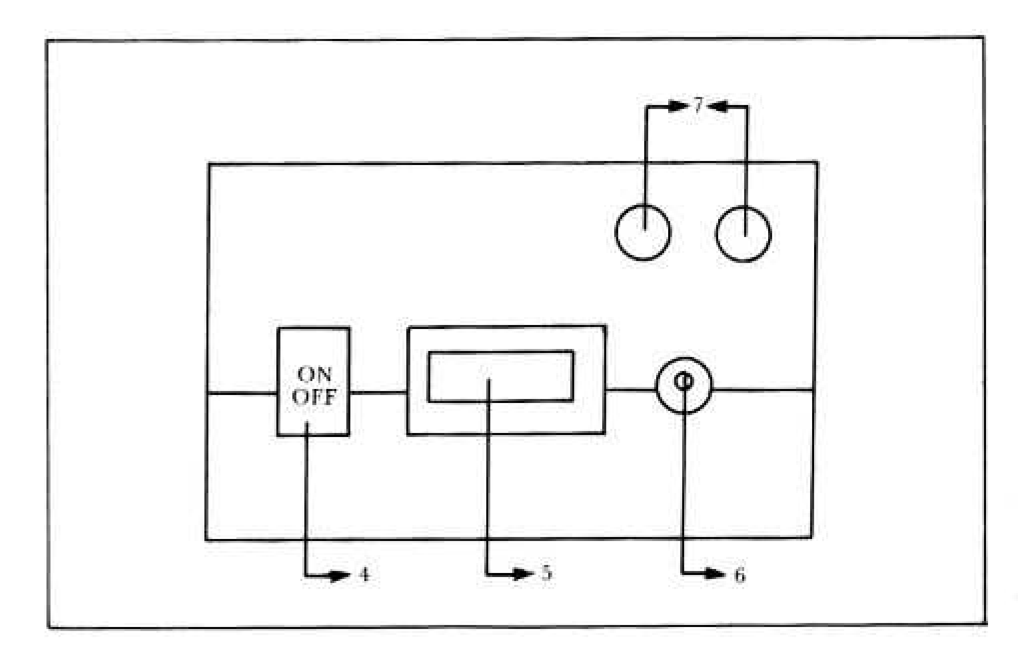


Fig. A.3. Vista posterior. 4. Interruptor. 5. Conexión a la red. 6. Fusible. 7. Conexiones en serie.

Coloque la unidad de discos cerca del C-64 y vamos a comenzar a instalarlo. En primer lugar, desconecte el Commodore. Tome el cable negro y conecte una de las clavijas terminales a una de las conexiones serie que hay en la parte posterior de la unidad de discos (ver Fig. A.3), y el otro extremo, en la conexión en serie del Commodore 64 (ver Fig. 1.3). Compruebe que el interruptor de la unidad de discos (ver Fig. A.3) está en posición de desconectado (OFF) y utilice el cable gris para conectar la unidad al enchufe de la pared. Antes de activar el interruptor compruebe que no hay ningún disquete dentro de la unidad y pulse el interruptor de la unidad de discos para activarlo (luz verde encendida). Pulse ahora el interruptor (ON/OFF) del C-64 y ya podemos empezar a trabajar. Recuerde esta secuencia de conexión y que siempre el último interruptor a pulsar es el del Commodore.

Habrá observado que en la parte posterior de la unidad de discos aparecen dos enchufes para conexión en serie. Uno lo tiene ocupado para unir el Commodore con la unidad de discos, el otro es para poder conectar, a esta unidad de discos, otra unidad de discos o una impresora. Siguiendo esta secuencia podemos llegar a tener conectadas hasta cinco unidades de disco y una impresora.

El disquete

Es un disco pequeño (de cinco pulgadas y cuarto; 5¼) construido con un material plástico flexible y cubierto por una superficie magnética que es donde se grabarán los datos. Viene dentro de una funda protectora, generalmente negra, y colocado dentro de un sobre del que es fácil extraerlo. En las figuras A.4, A.5 y A.6 se exponen, respectivamente, la forma de introducir el disquete, la estructura interna del disquete y las especificaciones técnicas del disquete.

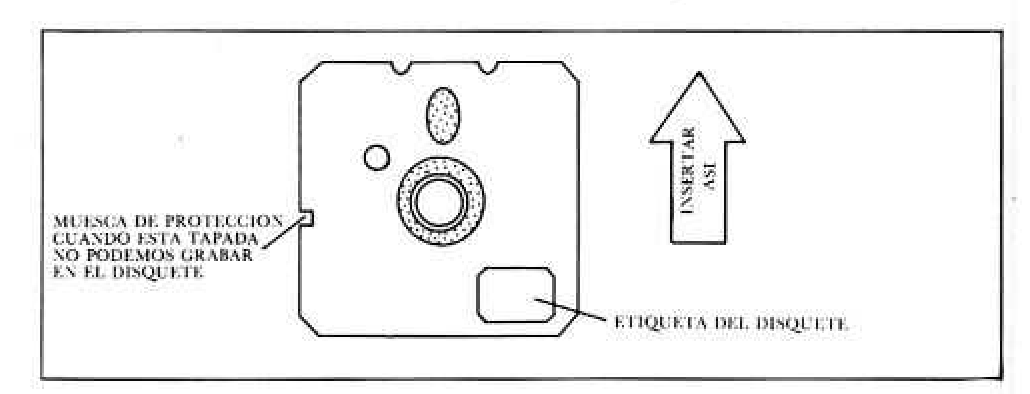


Fig. A.4. Posición para insertar el disquete.

Antes de insertar el disquete compruebe que la luz verde está encendida y la luz roja apagada. Presione la placa que tapa la boca de la unidad de discos y ésta se desplazará automáticamente hacia arriba. Tome el disquete (sáquelo del sobre, pero no de la funda) con la etiqueta hacia arriba e introdúzcalo con suavidad por la boca de la unidad. Si encuentra alguna resistencia, no empuje, vuelva a sacar y a introducir el disquete hasta que la operación se realice sin dificultad. Una vez dentro el disquete, empuje hacia abajo la placa de la boca y ya está preparado para utilizarlo.

Cuando utilice la unidad de discos se encenderá la luz roja. Si la operación se efectúa correctamente se apagará la luz roja, y si no desea utilizar más el disquete ya puede sacarlo. Si la operación ha sido incorrecta, la luz roja quedará intermitente. Realice la operación correctamente y esta intermitencia desaparecerá. No introduzca ni saque el disquete con la luz roja encendida o en intermitencia, esto puede dañar su contenido.

Para sacar el disquete presione hacia dentro la placa que tapa la boca de la unidad de discos, y un mecanismo automático hará que parte del disquete salga fuera para que pueda retirarlo cómodamente.

Observe, en la figura A.4, la muesca de protección situada en el borde izquierdo. Generalmente en un disquete podemos grabar y leer información. Si queremos proteger un disquete, de modo que se pueda leer la información contenida en él, pero que no puedan grabar, no vaya a ser que estropeen nuestros programas, basta con tapar la muesca de protección con un trozo de cinta plástica o de papel adhesivo.

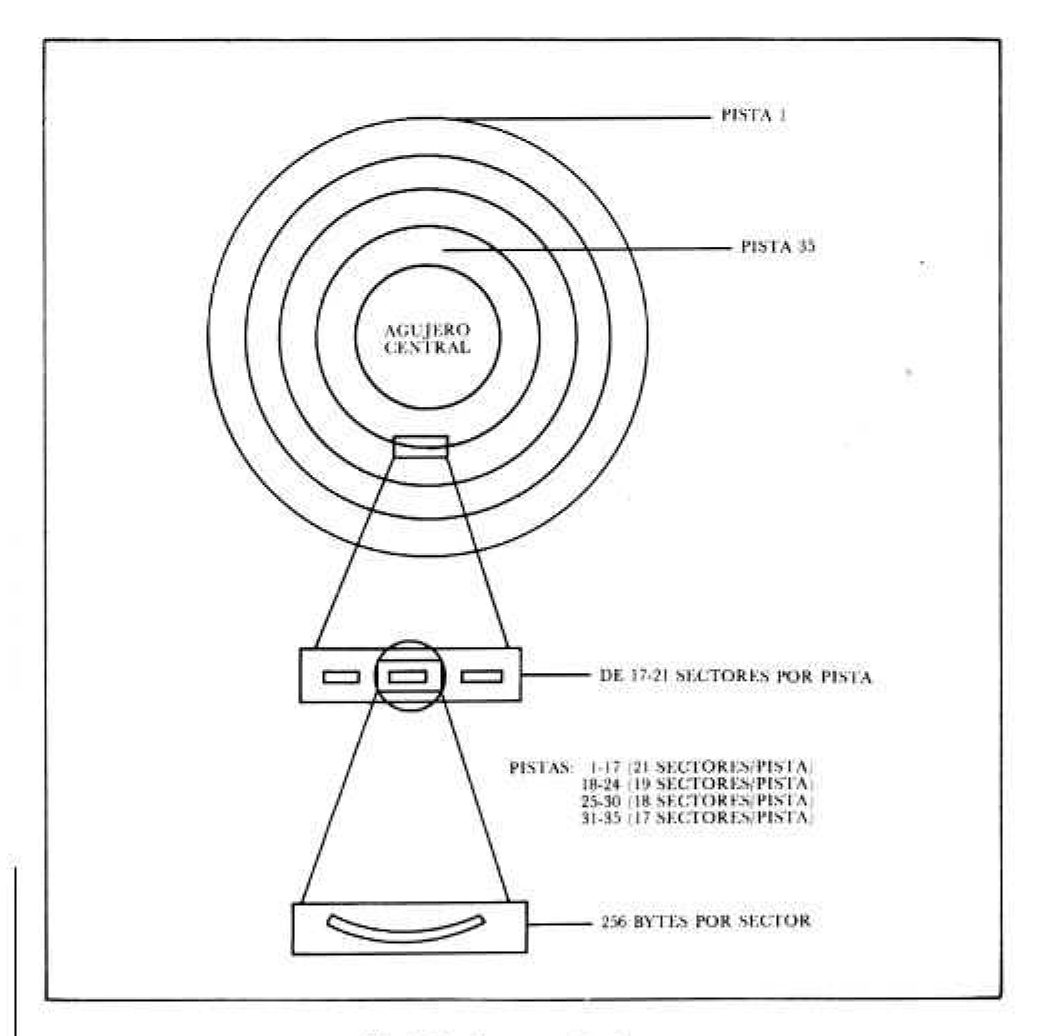


Fig. A.5. Estructura del disquete.

Almacenamiento:	
Capacidad total	174848 bytes por disquetes
Número de nombres de programas	141 por disquete
Sectores por pista	17-21
Bytes por sector	256
Número de pistas	35
Número de sectores	683 (644 bloques libres)
Memoria de la unidad de discos (Buffer)	2 K RAM
Tamaño de la unidad de discos:	
Altura	97 mm
Anchura	200 mm
Profundidad	374 mm
Condiciones eléctricas	
Voltaje	100, 120, 220, ó 240 V
Frecuencia	50 ó 60 Hz
Potencia	25 W
Características del disquete:	
Tamaño	5¼ pulgadas
Densidad	Simple cara, simple densidad

Fig. A.6. Características técnicas del disquete y de la unidad de discos.

En la figura A.6 se trata de ilustrar la estructura del disquete. Está formado por una serie de pistas (35) cada una de las cuales está dividida en sectores (entre 17 y 21 dependiendo de la pista). Estos disquetes son de simple cara, simple densidad, sin embargo, puede utilizar disquetes de simple cara, doble densidad, o de doble cara, doble densidad.

Todo depende del precio y la accesibilidad. En un disquete, podemos grabar muchos programas, pero el directorio sólo puede guardar hasta 141 nombres de programas o ficheros de datos.

En este apéndice vamos a ver todo lo relacionado con una unidad de discos, con ficheros de programas y con ficheros secuenciales de datos. Quizá eche en falta la explicación del uso de más de un disquete y de los ficheros de acceso aleatorio, pero ambas cosas exceden el nivel de este libro.

Ficheros de programas

Lo primero que desearemos hacer con nuestro disquete es ver qué programas contiene o grabar en ellos los programas que vayamos creando. Claro que, antes de grabar un programa, tenemos que inicializar el disquete, y esto lo veremos en la sección siguiente.

Reproducción de programas

Para cargar en el C-64 los programas que tengamos grabados en el disquete utilizaremos el comando:

LOAD "«nombre del programa»", «núm. del dispositivo», «núm. de comando»

Nombre del programa: Una constante o variable alfanumérica que contenga no más de 16 caracteres. Así, nombres válidos son «PRUEBA», «PROGRAMA 14», A$, OTROS$.

Núm. del dispositivo: Si estamos utilizando una sola unidad de disquete, este número es el *8*. Si no ponemos este número, el C-64 intentará cargar el programa de la cinta.

Núm. de comando: Si omitimos este número o ponemos un 0, el programa se cargará en la memoria normalmente, esto es, desde la primera dirección de memoria disponible para programas en BASIC. Si ponemos un 1, el programa se cargará en las mismas direcciones de memoria que ocupaba cuando se grabó.

Por ejemplo, para cargar el programa «PRACTICA» utilizaríamos el comando:

LOAD "PRACTICA", 8

Pulse RETURN, y si el programa «PRACTICA» estaba grabado en el disquete verá aparecer en la pantalla la secuencia:

```
SEARCHING FOR PRACTICA
LOADING
READY
```

Y ahora ya puede teclear RUN y pulsar RETURN para ejecutar el programa. Si el programa no estaba en este disquete aparecerá la secuencia:

```
SEARCHING FOR PRACTICA
? FILE NOT FOUND ERROR
READY
```

Y la luz roja de la unidad de discos quedará intermitente.

Reproducción del directorio

Es decir, puede que no supiéramos cuál era el nombre del programa o nos hayamos equivocado al teclearlo. Para ver los nombres de programas que tenemos cargados en un disquete, introduzca el comando:

LOAD "$", 8

Y verá aparecer los mensajes:

```
SEARCHING FOR $
LOADING
READY
```

Teclee LIST, pulse RETURN y en la pantalla aparecerá el listado de los nombres de los ficheros grabados en el disquete.

Grabación de programas

Para grabar programas en el disquete, introduzca el comando:

SAVE "«nombre del programa»", 8

La unidad de discos mirará primero si todavía cabe un nombre de programa más en el directorio, después si hay suficientes sectores para almacenar el programa, y si todas estas comprobaciones son satisfactorias, su programa se grabará en el disquete.

Verificación de la grabación

Para verificar si la grabación ha sido correcta, introduzca el comando:

VERIFY "«nombre del programa»", 8

Y así comprobará si el programa cuyo nombre es «nombre del programa» coincide con el programa que actualmente hay en la memoria del C-64. Si la verificación no finaliza con éxito tendrá que comprobar si el nombre del programa indicado es correcto, y si así es tendrá que volver a grabar el programa. Puede que haya intentado grabar sin un disquete en la unidad de discos.

Hablando con la unidad de discos

Aparte de estas acciones sencillas, reproducir, grabar, verificar, para aprovechar las posibilidades del disquete tendremos que aprender una nueva instrucción (bueno, ya la hemos visto en el capítulo 3, pero no está de más recordarla).

OPEN «n.f», «n.d», «canal», "«texto»"

n.f: Es el número de fichero. Número que utilizamos para acceder al fichero, puede ser cualquier número del 1 al 255, pero es peligroso utilizar números mayores que el 127, pues la sentencia PRINT # genera un salto de línea adicional a la generada por el carácter del RETURN.

n.d.: Número del dispositivo. En el caso de una unidad de disquetes es el 8.

Canal: Es el número de canal que utilizamos para acceder al disco. Los números Ø y 1 se utilizan para reproducir (LOAD) y grabar (SAVE), respectivamente. Los números del 2 al 14 son los que emplearemos para los ficheros de datos. El número 15 es el dedicado al *canal de comandos*.

Texto: Es un texto que imprimiremos en el fichero.

OPEN es la instrucción que nos permitirá dar órdenes a la unidad de discos y manejar fácilmente el contenido del disquete que tengamos en ella. Es difícil pensar en algún ejemplo, así que lo mejor es comenzar a utilizarla rápidamente.

Recuerde que de los tres números que siguen al OPEN, el primero puede ser cualquier número del 1 al 255. El segundo será un 8, si estamos utilizando una sola unidad de discos, y el tercero, el número de canal, dependerá de la operación que estemos realizando.

El canal de comandos

Como acabamos de ver, el número que corresponde al canal de comandos (tercer número tras el OPEN) es el 15. ¿Y qué comandos directos le podemos dar a la unidad de discos? Pues le podemos decir que copie un programa, que borre otro, que le cambie el nombre a un fichero y muchas cosas más que veremos a continuación.

Formateando un disquete

Recién comprado un disquete, y antes de poder utilizarlo, para grabar algo, tenemos que formatearlo. Es algo así como un reconocimiento, por parte de la unidad de discos, del estado del disquete, se marca cada bloque y se crea el directorio. Tenga cuidado de no formatear un disquete en el que tenga grabado algo que sea importante, pues en este proceso borrará todos los ficheros que tenga grabados. El comando tiene la estructura siguiente:

OPEN 1,8,15

PRINT #1, "NEW: nombre del disco, ID"

NEW: Es el comando (nuevo).
Nombre del disco: 16 caracteres como máximo.
ID: Número de identificación. Como máximo dos caracteres.

Observe que detrás de PRINT va el símbolo de almohadilla y detrás de éste va el mismo número que hayamos puesto en primer lugar tras el OPEN.

La palabra NEW, la podemos abreviar por la N, quedando la segunda línea en la forma:

PRINT#1, "N: nombre del disco, ID"

Si tenemos un disquete ya formateado y lo único que deseamos es cambiarle el nombre y borrar todos los datos que hay en él utilizaríamos:

PRINT#1, "N: nombre del disco"

Este proceso es mucho más rápido que formatear el disco de nuevo y conservamos el mismo número de identificación.

Fíjese en que hemos separado el comando OPEN del comando PRINT#, en realidad esto no es necesario. Como veíamos en el apartado anterior, podemos ponerlo todo seguido en la forma:

```
OPEN, 1,8,15, "N: nombre del disco, ID"
```

Esto es aplicable a este caso y a los que vamos a ver a continuación, sin embargo recuerde que cuando estamos trabajando con el canal de comandos, la longitud de la orden no puede sobrepasar la longitud de una línea, esto es, cuarenta caracteres.

Inicialización

Este comando sirve para volver la unidad de discos a su estado original, siempre que ocurra una situación de error. Su formato es:

```
OPEN 4,8,15, "INITIALIZE"
```

ó bien:

```
OPEN 4,8,15, "I"
```

Copia de ficheros

Este comando nos permite copiar un fichero (bien sea un programa o un fichero de datos) con otro nombre en el mismo disquete. Esto es, tenemos un programa que se llama «Viejo nombre» y le decimos a la unidad de discos que nos cree una copia de este fichero con el nombre «Nuevo nombre» mediante el comando:

```
OPEN 6,8,15
PRINT#6, "COPY: Nuevo nombre=Viejo nombre"
```

Y ahora tenemos dos programas iguales: uno en «Viejo nombre» y otro en «Nuevo nombre». Como en todos los comandos de esta sección, la palabra COPY la podemos abreviar por su primera letra, en este caso la C, quedando el comando en la forma:

```
OPEN 6,8,15, "C: Nuevo nombre=Viejo nombre"
```

Este comando nos sirve también para unir, o concatenar, dos ficheros o para crear un nuevo fichero. Por ejemplo, suponga que queremos unir la Subrutina 1 y la Subrutina 2 para crear el programa Nuevo prog.; en este caso utilizaremos:

```
OPEN 3,8,15
PRINT#3, "C: Nuevo prog.=Subrutina 1, Subrutina 2"
```

Observe que con ello no borramos ni cambiamos los nombres de los programas Subrutina 1 y Subrutina 2. Simplemente creamos un programa "Nuevo prog." con una copia del contenido de los dos anteriores.

Cambio del nombre de un fichero

También existe un comando que nos permite cambiar directamente el nombre de un fichero (renombrar). Suponga que tenemos un programa llamado «Viejo nombre» y ahora queremos llamarle "Nuevo nombre". Utilice el formato:

```
OPEN 15,8,15
PRINT#15, "RENAME: Nuevo nombre=Viejo nombre"
```

Si ahora cargamos el directorio (LOAD "$", 8), veremos que el programa "Viejo nombre" ya no existe y que en su lugar ha aparecido otro llamado "Nuevo nombre".

Como siempre, el comando RENAME lo podemos abreviar mediante su primera letra: R.

```
OPEN 15,8,15
PRINT# 15, "R: Nuevo nombre=Viejo nombre"
```

Borrado de ficheros

Para borrar un fichero del disquete, utilizaremos el comando SCRATCH. Por ejemplo, para borrar el fichero cuyo nombre es «NOMBRE DE FICHERO», utilizaríamos el formato:

```
OPEN 1,8,15
PRINT#1, "SCRATCH: NOMBRE DEL FICHERO"
```

O utilizando la abreviatura:

```
OPEN 1,8,15
PRINT#1, "S: NOMBRE DEL FICHERO"
```

Validación

Tras utilizar durante largo tiempo un disquete, ocurre que, tras crear y borrar muchos programas, dejar ficheros de datos sin cerrar (procure que esto no

le ocurra muy a menudo), tendremos el disquete muy desorganizado, con muchos bloques sueltos sin utilizar y, aunque sean muchos, son poco utilizables, pues no están agrupados. Para arreglar esta situación y volver a organizar el disquete utilizamos el comando VALIDATE.

```
OPEN 1,8,15, "V"
```

o bien:

```
OPEN, 1,8,15, "VALIDATE"
```

CLOSE

Acostúmbrese a utilizar esta instrucción después de haber finalizado las operaciones a realizar tras el OPEN inicial. Sobre todo en ficheros de datos será muy importante, pues antes de enviar los datos al disquete se van almacenando en cierta parte de la memoria y, si no utiliza el comando CLOSE, se quedarán allí sin ser enviados al fichero en que deseábamos que estuviesen.

Ficheros de datos secuenciales

Muchas de las advertencias en la forma de manejo de este tipo de ficheros son similares para el disquete y la Datassette. Así pues, sería una buena medida que releyera aquella sección antes de proseguir. Vamos a ver aquí las principales diferencias, y sobre todo usted comprobará la gran diferencia: la velocidad.

Al igual que en la Datassette, antes de poder utilizar un fichero de datos tenemos que abrirlo (OPEN); el formato de la instrucción es:

```
OPEN «n. f», «n.d», «canal», "Ø: nombre, SEQ, W"
```

n.f: Número del fichero, del 1 al 255.
n.d: Número del dispositivo, en caso de un solo drive, el 8.
canal: Para ficheros de datos, del 2-14 (recuerde que el Ø y el 1 se reservan para LOAD y SAVE y que el 15 es el del canal de comandos).
nombre: nombre que queramos ponerle al fichero.
SEQ: Indica que el fichero es secuencial.
W: Indica que lo abrimos para escribir (WRITE). Si lo queremos abrir para leer su contenido tendremos que poner una R (READ).

Por ejemplo:

```
OPEN 4, 8, 3, "Ø: PRUEBA, SEQ, W"
```

Un fichero que ya existe sólo lo podemos abrir para leerlo y no para escribirlo. Si intentamos abrirlo para escribir obtendremos un error y la luz roja de la unidad de discos quedará parpadeando. Si queremos escribir en un fichero que ya existe podemos introducir el comando:

```
OPEN 4, 8, 3, "@Ø: PRUEBA, SEQ, W"
```

Pero recuerde que así borrará los datos que ya existían, pues escribirá sobre ellos.

No es necesario que el nombre del fichero esté determinado en la instrucción de apertura, OPEN. Es decir, podemos utilizar una variable de cadena dentro de un programa y poner así diversos nombres a los diferentes ficheros que creemos con ese programa. Para esto podrían valer las líneas:

```
1ØØØ INPUT "NOMBRE DEL FICHERO"; NF$
1Ø1Ø OPEN 3, 8, 5, "Ø:" + NF$ + ",SEQ, W"
```

El resto de las sentencias para manejar ficheros de datos secuenciales, ya las vimos con detenimiento al hablar de ficheros en Datassette. Son los comandos: CLOSE, para cerrar el fichero; PRINT#, para enviar datos al fichero; INPUT# y GET#, para leer datos de los ficheros.

Acuérdese de cerrar (CLOSE) los ficheros después de haber terminado de utilizarlos, y sobre todo, si está escribiendo en ellos. Si no cierra el fichero perderá todos los datos que estén esperando en la memoria del ordenador a ser transmitidos.

Recuerde también que con un INPUT# no puede leer cadenas mayores de 8Ø caracteres; para cadenas mayores tendrá que utilizar el GET#, que lee carácter a carácter.

Al igual que hacíamos cuando trabajábamos con la Datassette, es conveniente comprobar el valor de la variable ST. ST es una variable del sistema cuyo contenido depende del resultado de la última operación de lectura. Los posibles valores que puede tomar, cuando utilizamos la unidad de discos, se muestran en la figura A.7.

Valor de ST	*Significado*
1	El dispositivo receptor no está disponible.
2	El dispositivo transmisor no está disponible.
4,8,16,32	Ninguno.
64	Fin de fichero.
128	La unidad de discos no está conectada.

Fig. A.7. Posibles valores de ST.

Vamos a ver ahora un ejercicio sencillo que nos permita repasar todo lo que hemos visto en esta sección. Se trata de crear un fichero con los nombres de nuestros amigos. El programa nos saluda (líneas 6Ø-1ØØ) informándonos de lo que debemos de hacer: para introducir un nombre, lo tecleamos cuando nos lo pida y pulsamos la tecla RETURN. Si ya no deseamos introducir más nombres, pulsamos la tecla X y después RETURN. En la línea 5Ø se abre un fichero de datos con el nombre de «NOMBRES 1», y la línea 16Ø se encarga de escribir en él, uno tras otro, los nombres que pongamos. Observe que la luz roja de la uni-

dad de discos no se enciende tras cada nombre. Sólo se enciende si ya llevamos muchos nombres, o si se ejecuta la línea 180, pues los nombres se van guardando en una zona de memoria, y cuando ésta se llena se transmiten todos juntos a una gran velocidad. Si no llegamos a llenarla, o si al final quedan algunos nombres más, la sentencia CLOSE se encarga de terminar de escribir estos datos en el fichero y de cerrarlo adecuadamente.

```
10 REM*************************
20 REM*CREACION DE UN FICHERO*
30 REM*************************
40 PRINT"♥"
50 OPEN 2,8,4,"0:NOMBRES1,SEQ,W"
60 PRINT:PRINT:PRINT TAB(16);"NOMBRES"
65 PRINT
70 PRINT"ESTE PROGRAMA ALMACENA NOMBRES"
80 PRINT:PRINT"EN EL DISKETTE. POR FAVOR"
90 PRINT:PRINT"ESCRIBA CADA NOMBRE CUANDO "
100 PRINT:PRINT"SE LE PIDA."
110 PRINT:PRINT"INTRODUZCA LA X PARA TERMINAR"
120 FOR I=1 TO 5000:NEXT
130 PRINT"♥"
140 INPUT"NOMBRE";N$
150 IF N$="X" THEN 180
160 PRINT#2,N$
170 GOTO 130
180 CLOSE 2
190 REM*****************************
200 REM*LECTURA DEL FICHERO CREADO*
210 REM*****************************
220 PRINT"♥"
230 PRINT:PRINT:PRINT
240 PRINT"SI DESEA LEER EL FICHERO CREADO"
250 PRINT:PRINT"PULSE LA LETRA L"
260 GET A$:IF A$<>"L" THEN 260
270 PRINT"♥"
280 OPEN 4,8,4,"0:NOMBRES1,SEQ,R"
290 INPUT#4,J$
300 PRINT J$
310 IF ST=64 THEN 330
320 GOTO 290
330 CLOSE 4
340 PRINT:PRINT:PRINT TAB(16);"SE ACABO"
```

Después, las líneas 240-250 nos indican que, si deseamos ver el fichero, pulsemos la letra L. Si así lo hace, la línea 290 se encarga de leer los nombres guardados en el fichero y la línea 300 de imprimirlos en la pantalla, hasta que se detecte, mediante ST, que hemos llegado al final del fichero.

Anímese y cree un fichero con los nombres, direcciones y teléfonos de sus amigos.

La impresora

Para conectarla tenemos que utilizar el mismo enchufe, conexión en serie, que para la unidad de discos. Si ya tiene conectada la unidad de discos, observe que en la parte posterior de esta última existen dos conectores en serie idénticos. Uno de ellos estará ocupado por la conexión entre la unidad de discos y el C-64. Utilice el otro para introducir la clavija de la impresora.

La impresión en papel, o el envío de datos para que se escriban en el papel, se trata como si fuera un fichero de datos al que hemos de dirigir éstos. Así, tendremos que abrir un fichero mediante la intrucción:

OPEN «n.f», «n.d», «canal»

n.f: Número del fichero, del 0 al 225.
n.d: Número del dispositivo. En el caso de la impresora podemos utilizar el 4 ó el 5. Este número se selecciona utilizando el interruptor T-4-5 que hay en la parte posterior de la impresora.
Canal: En general no se utiliza. Sólo se emplea para indicar que queremos que escriba en «Modalidad minúsculas» en lugar de «Modalidad mayúsculas». En caso de que deseemos esto tendremos que poner un 7.

Por ejemplo, para escribir «Esta es mi impresora», introduzca los comandos.

```
OPEN 1,4
PRINT#1, "ESTA ES MI IMPRESORA"
```

Siempre debemos recordar que tras utilizar un fichero hemos de cerrarlo. En el ejemplo anterior utilizaríamos el comando:

```
CLOSE 1
```

La estructura general de la sentencia CLOSE es:

CLOSE «número del fichero»

Todo lo que generalmente vemos aparecer en la pantalla lo podemos enviar a la impresora. Para ello tenemos que utilizar la instrucción CMD. Introduzca las líneas:

```
OPEN 5,4
CMD 5
```

Y observará que todo lo que normalmente aparecería en la pantalla sale ahora por la impresora. Incluso los mensajes de error y el mensaje READY.

Para salir de esta situación tenemos varias opciones. La primera es pulsar con decisión y al mismo tiempo las teclas RUN/STOP y RESTORE. La se-

gunda es volver a utilizar la instrucción CMD para direccionar de nuevo la salida a la pantalla: CMD 3, pues 3 es el número de dispositivo correspondiente a la pantalla. Y la tercera es poner un PRINT#«n.f» vacío que imprimirá un retorno de carro y liberará a la impresora (en el ejemplo anterior PRINT#5).

Cuando utilice la impresora recuerde:

a) Que la acción del separador «;», en los PRINT#, es la misma que cuando utiliza la pantalla. Esto es, se escribe cada dato a continuación del otro. (Las variables numéricas van precedidas de un espacio en blanco.)
b) Sin embargo, la acción del separador «,» varía un poco, pues su efecto es colocar once espacios en blanco entre cada par de datos.
c) Las acciones de las funciones TAB y SPC son idénticas y coinciden con el efecto del SPC en la pantalla. Sin embargo, no pueden ir directamente detrás del PRINT# «n.f», sino que hemos de separarlos, aunque sea mediante una cadena vacía, así, por ejemplo:

```
OPEN 3, 4
PRINT#3, "SPC (7)"; "HOLA"
CLOSE 3
```

Apéndice B: Lista de comandos

CLOSE

CLOSE se utiliza para cerrar los ficheros abiertos previamente con comandos OPEN. Cada fichero debe cerrarse antes de la terminación del programa para garantizar su integridad. No puede usarse con un número de fichero para el que no se ha ejecutado un OPEN.

SINTAXIS

CLOSE nf
Siendo nf el número del fichero.

Ejemplo

CLOSE 5
Cierra el fichero número 5

CLR

CLR es la abreviatura de CLEAR, que significa borrar, se utilizará para poner a cero todas las variables numéricas y asignar contenido nulo a las alfanuméricas sin afectar al programa. Cierra todos los ficheros que estén abiertos.

SINTAXIS

CLR

Ejemplo

1000 CLR

CMD

CMD se utiliza para enviar a un periférico distinto de la pantalla la información que normalmente sale a ésta, por ejemplo, con PRINT o LIST.

SINTAXIS

CMD nf
Siendo nf el número de fichero.

Ejemplo

```
90 OPEN 1,4
100 CMD 1
```

Con la línea 100 dirigimos la información de salida al fichero número 1, previamente asociado al periférico número 4, que es la impresora (línea 90).

CONT

CONT tecleado como comando directo continúa la ejecución de un programa detenido con STOP o END en el mismo lugar en que se encontraba cuando se produjo la interrupción, siempre y cuando no se haya modificado en nada el programa, ni añadido líneas nuevas, ni tan siquiera se haya pulsado RETURN para cambiar el cursor a otra línea.

SINTAXIS

CONT

DATA

Con DATA guardamos datos numéricos o alfanuméricos separados por comas, que serán asignados a variables al leerlos con instrucciones READ.

SINTAXIS

DATA constante [constante, constante ... constante]

Ejemplo

```
30 DATA ESTER, ALFREDO
40 DATA 15, 16, 6.5, PILAR
50 DATA 19, 20, —1. ENRIQUE, 6, LUIS
```

DEF FN

DEF FN nos permite trabajar con funciones definidas por el usuario. Conseguimos así ahorrar memoria cuando una fórmula debe usarse varias veces a lo largo de un programa. No puede usarse como comando directo.

SINTAXIS

DEF FN Nf (argumento) = «expresión»
Siendo Nf el nombre de la función.

Ejemplo

```
10 DEF FN B(Y) = 74*(34.75↑2*Y/3)
20 PRINT FN B(4)
```

DIM

DIM se utiliza para reservar memoria para variables con índices. No es necesario si se trabaja con índices menores de 11. Cada variable sólo puede dimensionarse una vez en el programa, salvo que se use CLR.

Sintaxis

DIM NOMBRE DE VARIABLE (argumento numérico)

Ejemplo

```
10 DIM A (5)
20 DIM A$ (44, 3)
```

END

Cuando un programa encuentra una instrucción END detiene su ejecución, lo cual permite que tenga más de una posibilidad de finalización distinta que el fin físico del programa. La ejecución puede reanudarse con CONT.

SINTAXIS

END

Ejemplo

```
200 END
```

FOR-TO-NEXT

Las instruciones FOR TO y NEXT permiten repetir una serie de acciones las veces deseadas.

SINTAXIS

```
FOR «NOMBRE DE VARIABLE» = Vi TO Vf STEP «INCREMENTO»
________ }
________ } instruciones a repetir.
________ }
NEXT «NOMBRE DE VARIABLE»
```

Siendo Vi = valor inicial
Vf = valor final

Es opcional poner o no el nombre de la variable a continuación del NEXT. Tampoco es imprescindible STEP, ya que el BASIC asume incremento 1 por defecto.

Ejemplo

```
10 FOR I = 1 TO 15
20 PRINT "PACO"
30 NEXT I
```

GET

GET nos permite introducir un carácter numérico o alfanumérico desde el teclado, pero, a diferencia de INPUT, no hay que pulsar RETURN, con sólo pulsar el carácter, éste es recibido. Mientras no se pulse ningún carácter devolverá 0 o la cadena vacía, dependiendo de que el nombre de variable que le acompañe sea numérico o alfanumérico. Si se utiliza un nombre de variable numérico y se pulsa cualquier tecla que no sea un dígito aparecerá el mensaje SYNTAX ERROR.

SINTAXIS

GET NOMBRE DE VARIABLE [, NOMBRE DE VARIABLE..., NOMBRE DE VARIABLE]

Ejemplo

100 GET B$: IF B$ = "" THEN 100

Con esta línea de programa conseguimos que el programa esté detenido hasta que pulsemos una tecla.

GET#

GET# se utiliza después de haber abierto un fichero con la instrucción OPEN, para permitir la entrada de caracteres de uno en uno en forma similar a como funciona GET para el teclado.

SINTAXIS

GET# nf, NOMBRE DE VARIABLE [, NOMBRE DE VARIABLE..., NOMB. DE VARIABLE]

Siendo nf el número de fichero

GOSUB

La instrucción GOSUB envía el control del programa a la línea que se especifique a continuación y sigue ejecutando desde ella hasta que encuentre una instrucción RETURN. El programa retrocede entonces hasta la sentencia siguiente a la que contenía el GOSUB. Se utiliza cuando hay que hacer uso de una misma rutina en diferentes partes del programa.

SINTAXIS

GOSUB NUMERO DE LINEA

Ejemplo

100 GOSUB 1000

Esta línea ordena ejecutar la subrutina que comienza en la línea 1000

GOTO

La instrucción GOTO envía el control del programa a la línea que se especifique a continuación y sigue la ejecución a partir de ella.

SINTAXIS

GOTO NUMERO DE LINEA

Ejemplo

100 GOTO 250

La instrucción GOTO envía el control a la línea 250

IF THEN

IF... THEN permite tomar decisiones acerca de una situación. Si la condición entre el IF y el THEN es cierta, se ejecuta la acción indicada a continuación del THEN; si no lo es, no se ejecuta y el control pasa a la línea siguiente. La condición puede ser compuesta si se utilizan los operadores lógicos AND, OR y NOT.

SINTAXIS

IF «CONDICION» THEN «ACCION»

Ejemplo

```
100 IF A<B THEN PRINT "CIERTO"
500 IF X> Y AND Y> Z THEN C = C+1
```

INPUT

La instrucción INPUT permite introducir datos numéricos o alfanuméricos desde el teclado, según el tipo de las variables que le acompañan. Si se pone un mensaje, éste debe ir separado del primer nombre de la variable mediante un punto y coma, y a continuación, separados por comas, pueden ponerse más nombres de variables.

SINTAXIS

INPUT «MENSAJE»; VARIABLE [, VARIABLE,..., VARIABLE]

El mensaje es opcional

Ejemplo

```
10 INPUT "SU NOMBRE"; A$
20 INPUT "INTRODUZCA 3 NUMEROS ENTEROS"; A%, B%, C%
30 INPUT A, B
```

INPUT#

INPUT# funciona como INPUT, pero se utiliza para extraer datos de un dispositivo identificado mediante un número de fichero lógico.

SINTAXIS

INPUT# nf, VARIABLE [, VARIABLE,..., VARIABLE]
Siendo nf el número de fichero

Ejemplo

```
100 INPUT# 9, A
200 INPUT# 5, C$, H
```

LET

Let es la sentencia de asignación y se utiliza para asignar contenidos a variables, tanto numéricas como alfanuméricas. La palabra LET es opcional y normalmente se omite.

SINTAXIS

LET «NOMBRE DE VARIABLE» = «EXPRESION»

Ejemplo

```
10 LET A =6
50 LET A$ = "CASA 4"
90 A% = 56
```

LIST

Con la instrucción LIST conseguimos listar el programa que esté actualmente en la memoria del ordenador.

SINTAXIS

LIST
LIST NUMERO DE LINEA
LIST —NUMERO DE LINEA—
LIST NUMERO DE LINEA—
LIST NUMERO DE LINEA— NUMERO DE LINEA

Ejemplo

```
LIST 60
LIST- 80
```

LOAD

La instrucción LOAD sirve para cargar un programa desde un dispositivo a la memoria del ordenador.

SINTAXIS

LOAD ["NOMBRE" [, NUMERO DE DISPOSITIVO]]
Si se omite el número de dispositivo se carga el programa desde el Datasette.

Ejemplo

LOAD
Carga el primer programaf que se encuentre en la cassette
LOAD "PROG. 1", 8
Carga del disco el programa llamado PROG. 1

NEW

El comando NEW borra el programa que el ordenador tenga en ese momento en memoria. Todas las variables numéricas se inicializan a 0 y las alfanuméricas a cadena nula o vacía. Se utiliza como comando directo, pero introducido en una línea de programa su efecto es el mismo, borra el programa al llegar la ejecución a la línea que contiene el NEW.

SINTAXIS

NEW

ON-GOSUB

ON debe ir seguido de una fórmula que al evaluarse genera un número, y GOSUB lleva a continuación una lista de números de línea separados por comas. Si al evaluar la fórmula el resultado es 1, se ejecuta la rutina que comienza en el primer número de la lista; si es 2, la segunda, y así sucesivamente. Si es 0 o mayor que el número de líneas que hay detrás el control, pasa a la línea siguiente a la que contiene el ON-GOSUB.

SINTAXIS

ON «expresión» GOSUB NUMERO DE LINEA [, NUMERO DE LINEA,..., NUMERO DE LINEA]

Ejemplo

```
10 INPUT A
20 ON A GOSUB 100, 70, 90
```

ON-GOTO

Funciona exactamente igual que ON-GOSUB, con la diferencia de que no hay posibilidad de retorno de subrutina.

SINTAXIS

ON «expresión» GOTO número de línea [, número de línea,..., número de línea]

Ejemplo

```
10 INPUT A
20 ON A GOTO 100, 70, 90
```

OPEN

OPEN permite acceder a periféricos tales como la impresora, la unidad de discos, el cassette. Debe ir seguida del número de fichero, que debe estar comprendido entre Ø y 225; de otro más que indica el número de dispositivos, que será Ø para la pantalla, 1 para la cassette, 4 para la impresora y 8 para la unidad de disco. Separado por coma puede ir un tercer número, que indica para qué hemos abierto el fichero; los valores que puede tomar dependerán del dispositivo; este número es opcional. Y, por último, también de forma opcional y separado por coma, podemos poner el nombre del fichero.

Sintaxis

OPEN nf, nd [, ds, [f$]]

Siendo: nf Número de fichero.
nd Número de dispositivo.
ds Número que indica para qué abrimos el fichero.
f$ Nombre del fichero.

POKE

El comando POKE se utiliza para colocar datos en posiciones concretas de la memoria. Va seguido de dos números o expresiones: el primero indica la posición de la memoria y debe estar comprendido entre Ø y 65535, y el segundo indica lo que introducirá en la posición de memoria especificada y debe estar comprendido entre Ø y 255.

Sintaxis

POKE n_1, n_2

Siendo: n_1 Número que indica la posición de la memoria.
n_2 Número que indica lo que ha de colocarse en dicha posición.

Ejemplo

```
10 POKE 36888,8
```

PRINT

PRINT es la sentencia que se utiliza para imprimir en la pantalla números o cadenas. Utilizando los separadores punto y coma, y coma conseguimos con un solo PRINT imprimir varios datos.

Sintaxis

PRINT «LISTA DE IMPRESION»

Ejemplo

```
10 PRINT "HOLA", "PEPE"
20 PRINT 2*6↑4/2+1
30 PRINT "AREA ="; 12
```

PRINT#

PRINT# se utiliza para grabar en el fichero, cuyo número se indica a continuación, los datos que, separados por comas, se indiquen, ya sean números, cadenas o nombres de variables.

Sintaxis

PRINT# nf, «LISTA»
Donde nf es el número del fichero.

Ejemplo

```
100 PRINT# 4, B, C$, "MAÑANA"
```

Esta línea graba en el fichero 4 el valor de B, el de C$ y la cadena «MAÑANA»

READ

READ asigna los datos contenidos en sentencias DATA a las variables que, separadas por comas, le acompañan.

Sintaxis

READ NOMBRE DE VARIABLE [, NOMBRE DE VARIABLE,..., NOMBRE DE VARIABLE]

Ejemplo

```
10 DATA 1, 6, 9, 10, 5.3
20 READ A, B, C, D, E
100 READ C$, F$, H, F
120 DATA RUBIA, MORENA, 65
130 DATA —05
```

REM

REM permite introducir comentarios en un programa. Las instrucciones REM no afectan a la ejecución del programa. Puede ir seguida de cualquier grupo de caracteres sin necesidad de utilizar comillas, pero evite los caracteres gráficos.

Sintaxis

REM «Comentario»

Ejemplo

```
10 REM AREA DE UN CONO
500 REM SUBRUTINA DE ACTUALIZACION
```

RUN

El comando RUN consigue la ejecución del programa que se encuentre en

la memoria del ordenador. Tecleando RUN la ejecución del programa se produce desde la primera línea que exista; si a continuación se teclea un número de línea, la ejecución comenzará a partir de dicha línea.

SINTAXIS

RUN [NUMERO DE LINEA]

Ejemplo

```
RUN
RUN 2ØØ
```

SAVE

SAVE es el comando utilizado para grabar un programa, existente en la memoria del ordenador, en cinta o disquete.

SINTAXIS

SAVE "NOMBRE DEL PROGRAMA"	para cinta
SAVE "NOMBRE DEL PROGRAMA", 8	para disquete

Ejemplo

```
SAVE "PROG. 15"
SAVE "PROG. 6.7", 8
SAVE
```

STOP

El comando STOP puede incorporarse a un programa para detenerlo en algún punto distinto del final físico, pero produce el mensaje BREAK ERROR IN LINE «Núm. de línea». La ejecución puede reanudarse con CONT.

SINTAXIS

STOP

Ejemplo

```
1ØØ STOP
2ØØ IF A>B THEN STOP
```

VERIFY

El comando VERIFY se utiliza para comprobar si el programa que acabamos de grabar en una cinta o disquete coincide con el que está actualmente en memoria. La finalidad es asegurarnos de que el programa se ha grabado correctamente.

SINTAXIS

VERIFY "NOMBRE DEL PROGRAMA"	para cassette
VERIFY "NOMBRE DEL PROGRAMA", 8	para disquete

WAIT

WAIT sirve para suspender la ejecución de un programa hasta que el contenido de una posición de memoria tome un valor determinado.

SINTAXIS

WAIT n1, n2 [, n3]

Donde: n1 Número que indica la posición de la memoria a examinar.
n2 Operando para AND.
n3 Operando para OR exclusivo (es opcional).
n2 y n3 Deben estar comprendidos entre Ø y 225.

Funciona de la siguiente manera: se extrae el contenido de la dirección de memoria n1 y se le aplica la operación lógica OR exclusiva; con n3 se ha especificado este parámetro. Al resultado, o en su caso al valor original, se le aplica a continuación la operación lógica AND con n2. Si el resultado es cero se repiten las mismas operaciones, y en otro caso se ejecuta la sentencia siguiente.

Ejemplo

```
1Ø POKE 162, 76
2Ø POKE 161, Ø
3Ø WAIT 161, 1
```

Estas tres líneas producen una espera de tres segundos.

FUNCIONES BASIC

ABS

ABS devuelve el valor absoluto del número que se le proporciona como argumento.

SINTAXIS

ABS (argumento numérico)

Ejemplo

PRINT ABS (5)	Imprimirá 5.
PRINT ABS (—6)	Imprimirá 6.

ASC

ASC devuelve el código ASCII del primer carácter de la cadena usada como un argumento.

SINTAXIS

ASC (argumento alfanumérico)

Ejemplo

PRINT ASC («A»)	Imprimirá 65.
PRINT ASC («ANA»)	Imprimirá 65, ya que utiliza como argumento sólo el primer carácter de la cadena.

ATN

ATN devuelve el arcotangente expresado en radianes del argumento que se le proporcione.

SINTAXIS

ATN (argumento numérico)

Ejemplo

PRINT ATN (—7)   Imprimirá —1.4288927

CHR$

CHR$ es la función inversa de ASC. Devuelve el carácter correspondiente al código ASCII que le introduzcamos. El argumento debe estar comprendido entre Ø y 225.

SINTAXIS

CHR$ (argumento numérico)

Ejemplo

PRINT CHR$ (65)   Devuelve el carácter A.

COS

COS devuelve el coseno del argumento numérico que le proporcionemos (el argumento hay que expresarlo en radianes).

SINTAXIS

COS (argumento numérico)

Ejemplo

PRINT COS (3Ø)   Imprimirá .154251445

EXP

EXP devuelve el valor de la constante e (2.71828183) elevado al número que se le dé como argumento, dicho número debe estar comprendido en el intervalo ± 88.Ø29691, sobrepasar por arriba el rango proporcionaría el mensaje OVER FLOW ERROR y por debajo imprimiría Ø

SINTAXIS

EXP (argumento numérico)

Ejemplo

PRINT EXP (2)	Imprimirá 7.389Ø561.
PRINT EXP (99.654321)	Imprimirá OVERFLOW ERROR.
PRINT EXP (—89.Ø39998)	Imprimirá Ø

FRE

FRE devuelve el número de bytes disponibles en la memoria, su argumento es simbólico y, por tanto, puede ser un número o una cadena. Es frecuente utilizarlo como comando directo.

SINTAXIS

FRE (argumento)

Ejemplo

Si nada más conectar el ordenador teclease PRINT FRE (2) obtendría —26627, pero el mismo valor conseguiría si teclease PRINT FRE (4) o PRINT FRE (A).

INT

INT devuelve la parte entera del número que se le dé como argumento. ¡Recuerde que el número que imprime es el entero menor!

SINTAXIS

INT (argumento numérico)

Ejemplo

PRINT INT (6.5)	Imprimirá 6.
PRINT INT (—2.5)	Imprimirá —3.

LEFT$

LEFT$ extrae el número de caracteres que se le indiquen de la parte izquierda de la cadena que se le proporcione como argumento.

SINTAXIS

LEFT$ (argumento alfanumérico, número)

Ejemplo

PRINT LEFT$ ("izquierda", 2)   Imprimirá iz.

LEN

LEN devuelve el número de caracteres que contiene la cadena que se le da como argumento.

SINTAXIS

LEN (argumento alfanumérico)

Ejemplo

PRINT LEN ("HOLA")   Imprimirá 4.
PRINT LEN (" HOLA ")   Imprimirá 6.

LOG

LOG devuelve el logaritmo en base e del argumento numérico que se le proporcione mientras éste no sea ni cero ni negativo, ya que entonces presentaría el mensaje ILLEGAL QUANTITY ERROR. Recuerde que si desea calcular un logaritmo en base decimal lo puede conseguir mediante LOG(X)/LOG(1Ø), donde X es el número cuyo logaritmo decimal desea calcular.

SINTAXIS

LOG (argumento numérico)

Ejemplo

PRINT LOG (1Ø)   Imprimirá 2.3Ø2585Ø9.

MID$

MID$ extrae n caracteres (n es un número a especificar como argumento) de la cadena indicada. También tenemos que concretar el carácter por el que ha de comenzar la extracción.

SINTAXIS

MID$ (argumento alfanumérico, m, n)
m Número del carácter en el que comienza la extracción.
n Número de caracteres a extraer.

Ejemplo

PRINT MID$ ("CADENA", 2, 3)   Imprimirá ADE.

PEEK

PEEK devuelve el contenido de una posición determinada de la memoria.

SINTAXIS

PEEK (dirección de memoria)

Ejemplo

PRINT PEEK (1)   Imprimirá 55.

POS

POS devuelve el número de columna en que se encuentra el cursor en la pantalla, su argumento es simbólico y, por tanto, puede ser numérico o alfanumérico.

SINTAXIS

POS (argumento)

Ejemplo

PRINT POS (5)   Imprimirá Ø
PRINT "HOLA"; POS (5)   Imprimirá HOLA 4.

RIGHT$

RIGHT$ extrae de una cadena, por ejemplo A$, el número de caracteres que le indiquemos, comenzando por la derecha.

SINTAXIS

RIGHT$ (argumento alfanumérico, número)

Ejemplo

PRINT RIGHT$ ("HOLA", 2)   Imprimirá "HOLA".

RND

RND genera números aleatorios comprendidos entre Ø y 1, tiene argumento numérico positivo. Si en una línea anterior asignamos a una variable RND (-argumento) conseguimos obtener siempre los mismos números.

SINTAXIS

RND (argumento)
RND (-argumento)

Ejemplo

```
1Ø X = RND (—1)
2Ø FOR I = 1 TO 3
3Ø PRINT RND (1)
4Ø NEXT I
```

Siempre que lo ejecutemos obtendremos la misma secuencia:

—32878082
—978964086
—895758909

SGN

SGN devuelve +1 cuando el número que se le da como argumento es positivo, —1 si es negativo y 0 si es cero.

Sintaxis

SGN (argumento numérico)

Ejemplo

PRINT SGN (—8)	Imprimirá —1.
PRINT SGN (0)	Imprimirá 0
PRINT SGN (76)	Imprimirá 1.

SIN

SIN devuelve el valor del seno del número que se le proporcione como argumento (el argumento ha de expresarse en radianes).

SINTAXIS

SIN (argumento numérico)

Ejemplo

PRINT SIN (45)   Imprimirá .80903524

SPC

SPC desplaza el cursor a la derecha el número de espacios que se indique. El texto no es modificado por el cursor.

SINTAXIS

SPC (argumento numérico)

Ejemplo

PRINT SPC (10): "HOLA"	Empezará a imprimir HOLA en la columna 10
PRINT "HOLA"; SPC (7); "HOLA"	Imprimirá HOLA, dejará siete espacios en blanco y volverá a imprimir HOLA.

SQR

SQR devuelve la raíz cuadrada del número positivo que se le dé como argumento. Si le proporcionamos un número negativo presentará el mensaje ? ILLEGAL QUANTITY ERROR.

SINTAXIS

SQR (argumento numérico)

Ejemplo

PRINT SQR (9)	Imprimirá 3.
PRINT SQR (26)	Imprimirá 5.09901951.
PRINT SQR (—1)	Imprimirá ? ILLEGAL QUANTITY ERROR.

ST

El valor de esta variable depende del resultado de la última operación de entrada/salida (ver la sección de ficheros de datos en Datassette y el apéndice de «Otros periféricos»). Es aconsejable comprobar su valor después de ejecutar cualquier sentencia que utilice los periféricos.

SINTAXIS

ST

Ejemplo

```
120 IF ST = 64 THEN PRINT "FIN DE FICHERO"
```

STR$

STR$ convierte en cadena el argumento numérico que se proporcione.

SINTAXIS

STR$ (argumento numérico)

Ejemplo

```
10 LET A$ = STR$ (19.8)
20 PRINT A$
```

SYS

SYS es una función del sistema que transfiere el control del programa a un subsistema independiente, su argumento debe ser un número decimal o una variable numérica cuyos rangos estén comprendidos entre 0 y 65535. Conseguimos con ella ejecutar el programa en lenguaje máquina a partir de la posición de memoria especificada.

SINTAXIS

SYS (dirección de memoria)

TAB

TAB mueve el cursor hasta el número de columna que se especifique como argumento.

SINTAXIS

TAB (argumento numérico)

Ejemplo

PRINT "HOLA"; TAB (7); "HOLA"	Imprimirá HOLA y la columna número 8 volverá a imprimir HOLA

TAN

TAN devuelve la tangente del argumento numérico (expresado en radianes) que se le proporcione.

SINTAXIS

TAN (argumento numérico)

Ejemplo

PRINT TAN (3Ø)	Imprimirá —6.4Ø533Ø91.

TI, TI$

TI y TI$ son variables del sistema y están relacionadas con el reloj de tiempo real que lleva incorporado el C-64. TI toma el valor Ø al encender el ordenador y su valor va aumentando con el tiempo que esté encendido, sólo puede volver a Ø cambiando el valor de TI$. TI$ es una cadena que consta de seis caracteres numéricos, los dos primeros, representan el número de horas, los dos siguientes el de minutos y los dos últimos el de segundos. Se le puede dar cualquier valor, siempre que los caracteres sean números, y quedará activado a partir de ese momento.

SINTAXIS

TI
TI$ ($H_1H_2M_1M_2S_1S_2$)

Ejemplo

```
1Ø INPUT "CUANTOS VALORES DE TI DESEA
OBTENER"; T
2Ø FOR I=1 TO T
3Ø PRINT TI
4Ø NEXT I
```

TI$="112Ø35"	Ajusta el reloj a las 11 horas 2Ø minutos 35 segundos (de la mañana).

USR

Es una función del sistema cuya misión es pasar un parámetro a una subrutina en lenguaje de máquina.

SINTAXIS

USR (argumento)

VAL

VAL devuelve el equivalente numérico de la cadena que se le proporcione como argumento, sólo si la cadena contiene números.

SINTAXIS

VAL (argumento alfanumérico)

Ejemplo

PRINT VAL ("123")	Imprimirá 123.
1Ø A$="456"	
2Ø PRINT VAL (A$)*2	Imprimirá 912.
PRINT VAL ("1A3")	Imprimirá 1.
PRINT VAL ("GHF")	Imprimirá Ø

ABREVIATURA DE LAS PALABRAS CLAVE DEL BASIC

Muchas de las palabras clave pueden abreviarse y esto nos ahorra tiempo cuando introducimos un programa en el ordenador. Si escribimos así nuestro programa, en apariencia será ilegible, lleno de caracteres extraños, pero al listarlo cada abreviatura se convierte en la palabra BASIC correspondiente. Por ejemplo, la abreviatura del PRINT es «?». Así, si teclea el siguiente programa:

1Ø? "ABREVIATURA"

y lo lista, observará:

1Ø PRINT "ABREVIATURA"

El resto de las abreviaturas se consiguen tecleando la primera o dos primeras letras de la palabra clave y a continuación, manteniendo pulsada la tecla SHIFT, pulsando la tecla correspondiente a la segunda o tercera letra de dicha palabra clave.

Por ejemplo, para conseguir abreviada la palabra CLOSE teclearemos CL y manteniendo pulsada la tecla SHIFT pulsamos la tecla de la letra O, y para conseguir abreviar la palabra clave FOR, teclearemos F y, manteniendo pulsada la tecla SHIFT, pulsaremos la tecla de la letra O. A continuación le presen-

tamos todas las posibles abreviaturas que puede conseguir con los símbolos que les corresponden. En la columna «Forma de abreviarlas», la letra situada dentro del recuadro indica que hay que mantener pulsada la tecla SHIFT al mismo tiempo:

Palabra clave	*Forma de abreviarla*	*Símbolo*
ABS	A[B]	A |
AND	A[N]	A/
ASC	A[S]	A/♥
ATN	A[T]	A |
CHR$	C[H]	C |
CLOSE	CL[O]	CL┌
CLR	C[L]	C └
CMD	C[M]	C ╲
CONT	C[O]	C┌
DATA	D[A]	D♠
DEF	D[E]	D—
DIM	D[I]	D╮
END	E[N]	E ╱
EXP	E[X]	E ♣
FOR	F[O]	E┌
FRE	F[R]	F—
GET	G[E]	G—
GOSUB	GO[S]	GO/♥
GOTO	G[O]	G┌
INPUT#	I[N]	I ╱
LET	L[E]	L—
LEFT$	LE[F]	LE—
LIST	L[I]	L╮
LOAD	L[O]	L┌
MID$	M[I]	M╮
NEXT	N[E]	N—
NOT	N[O]	N┌
OPEN	O[P]	O┌
PEEK	P[E]	P—
POKE	P[O]	P┌
PRINT	?	?
PRINT#	P[R]	P—
READ	R[E]	R—
RESTORE	RE[S]	RE/♥
RETURN	RE[T]	RE|
RIGHT$	R[I]	R╮
RND	R[N]	R ╱
RUN	R[U]	R╭
SAVE	S[A]	S♠
SGN	S[G]	S |
SIN	S[I]	S╮
SPC(	S[P]	S┌
SQR	S[Q]	S•
STEP	ST[E]	ST—
STOP	S[T]	S|
STR$	ST[R]	ST—
SYS	S[Y]	S|
TAB(	T[A]	T♠
THEN	T[H]	T|
USR	U[S]	U♥
VAL	V[A]	V♠
VERIFY	V[E]	V—
WAIT	W[A]	W♠

Las abreviaturas de las palabras clave TAB y SPC incluyen el paréntesis de apertura, es decir teclear S┐8) es equivalente a SPC (8), y teclear T♠8) equivale a TAB (8).

La forma de abreviar descrita es la más práctica, pero, en general, se puede abreviar una palabra clave tecleando sus primeras letras hasta la penúltima y, manteniendo pulsada la tecla SHIFT pulsando dicha penúltima letra, excepto en el caso de PRINT

Por ejemplo:

— Puede abreviar la palabra clave CLOSE de la forma descrita o tecleando CLO y, manteniendo pulsada la tecla SHIFT, pulsando la letra S.
— FOR sólo puede abreviarse de la forma descrita.

Apéndice C: Mensajes de error

Cuando el C-64 detecta un error, nos muestra un mensaje con el que nos comunica el tipo de error y, en el caso de que se trate de un programa, el número de línea en la que se ha producido el mismo.

El aspecto del mensaje es:

```
? «mensaje» ERROR IN LINE «numero de línea»
```

Obtener un mensaje de error al ejecutar un programa no quiere decir que ése sea el único error que contiene el programa, sino que la ejecución se detiene al encontrar el primero. Si lo corregimos y volvemos a realizar la ejecución puede que transcurra normalmente o que vuelva a presentar un nuevo mensaje de error.

A continuación le mostramos una lista completa de los mensajes de error que puede generar el BASIC del C-64, así como la descripción de sus posibles causas, pero tenga siempre en cuenta que hay un tipo de error que el ordenador nunca detecta: «los errores de lógica». No debe pensar que por el hecho de que un programa presente resultados, éste cumple su cometido, debe someterle a una sesión exhaustiva de pruebas en las que se le haga trabajar con todos los tipos de datos posibles para los que está previsto que deba funcionar.

LISTA DE MENSAJES DE ERROR EN ORDEN ALFABETICO

BAD SUBSCRIPT (Indice incorrecto)

Se produce cuando el programa hace referencia a un elemento de una matriz cuyo índice es mayor del especificado en la instrucción DIM. Obtendrá, por ejemplo, este error si dimensiona una variable DIM A$ (5Ø) e intenta trabajar luego con el elemento A$(51), también se producirá este mensaje cuando no se ha hecho dimensión previa para una variable y se intenta trabajar con un índice mayor a 1Ø.

BREAK (Rotura)

El mensaje BREAK ERROR se produce al pulsar la tecla RUN/STOP cuando se está realizando un acceso a la cinta.

CAN'T CONTINUE (No se puede continuar)

El comando CONT sólo puede usarse para continuar un programa después de haber pulsado la tecla RUN/STOP o de haberse ejecutado la sentencia STOP o la END y sólo si no se ha modificado en nada el programa. Si se modifica el programa después de la interrupción aparecerá este mensaje de error, así como si se utiliza CONT antes de empezar la ejecución de un programa con RUN.

DEVICE NOT PRESENT (El dispositivo no está presente)

Se produce este mensje cuando se ha utilizado alguna orden de entrada o salida (E/S) que hace referencia a un periférico que no está conectado, que funciona mal o cuya conexión no es correcta.

DIVISION BY ZERO (División por cero)

Se produce siempre que se intente realizar una división por cero, lo cual no es posible.

EXTRA IGNORED (Ignorados los datos extra)

Se produce cuando en la ejecución de una sentencia INPUT se introducen más datos de los solicitados, sólo acepta el número de datos que necesita, es decir, no se rompe la ejecución, pero presenta este mensaje. Por ejemplo, al ejecutarse la línea 3Ø INPUT A, B, C, el ordenador pide tres datos, si le tecleamos 4, 5, 6, 7, 8, 9, sólo acepta los tres primeros y presenta el mensaje.

FILE ALREADY EXISTS (El fichero ya existe)

Se produce cuando el nombre del fichero que se está copiando con el comando COPY ya existe en el disco de destino.

FILE DATA (Fichero con tipo de datos equivocado)

Se produce cuando el programa recibe datos alfanuméricos desde un fichero abierto cuando esperaba datos numéricos.

FILE NOT FOUND (Fichero no encontrado)

Se produce cuando intenta cargar (LOAD) un fichero en disco cuyo nombre no existe. Si está seguro de haber grabado el programa, compruebe el directorio por si acaso lo estuviese deletreando mal; si no es así, es posible que lo haya grabado en otro disco.

FILE NOT OPEN (Fichero no abierto)

Se produce cuando intenta hacer alguna operación (CLOSE, CMD, PRINT#, INPUT#, GET#) con algún fichero que no está previamente abierto.

FILE OPEN (Fichero abierto)

Se produce al intentar abrir (OPEN) un fichero utilizando como número el que ya tenía asignado algún otro fichero; la solución es fácil, dele otro número.

FORMULA TOO COMPLEX (Fórmula demasiado compleja)

Se produce cuando la expresión a evaluar es demasiado complicada. Se puede solucionar separando la expresión en partes, resolviendo cada parte y combinando después para obtener la solución.

ILLEGAL DEVICE NUMBER (Número ilegal de dispositivo)

Se produce siempre que intentamos acceder a un dispositivo en alguna forma prohibida; por ejemplo, daría este error la instrucción SAVE "PEPE", 3, porque no podemos salvar nada en el dispositivo número 3.

ILLEGAL DIRECT (En modo directo es ilegal)

Se produce cuando se utiliza como comando directo alguna instrucción que sólo puede usarse incorporada a un programa, éste es el caso de INPUT, INPUT#, DATA, DEF FN, GET, GET#.

ILLEGAL QUANTITY (Cantidad ilegal)

Se produce al usar como argumento de una instrucción algún dato que está fuera del rango permitido; por ejemplo, se produce si intenta guardar en variables numéricas enteras números más grandes o más pequeños que 32767 y —32768, respectivamente, o si utiliza como argumento de POKE un número mayor de 255 o menor que Ø etc.

LOAD (Carga)

Se produce cuando existe algún problema con un programa grabado en cinta.

MISSING FILE NAME (Nombre del fichero perdido)

Se produce al intentar acceder a un fichero sin especificar su nombre; por ejemplo, presentaría este mensaje la ejecución de la siguiente instrucción: SAVE " ", 8. El cassette es una excepción, ya que en él sí podemos acceder a un fichero sin especificar nombre.

NEXT WITHOUT FOR (NEXT sin FOR).

Se produce cuando el nombre de la variable de una instrucción NEXT no se corresponde con el utilizado en el FOR; por ejemplo, el siguiente programa mostraría este mensaje:

```
1Ø FOR I=1 TO 1ØØ
2Ø NEXT H
```

Se soluciona poniendo iguales los nombres de las variables o bien poniendo únicamente NEXT sin ningún nombre de variable detrás.
Este mensaje de error es típico de bucles FOR-NEXT mal anidados.

NOT INPUT FILE (El fichero no es de entrada)

Se produce cuando intentamos utilizar INPUT o GET en un fichero que se ha especificado que es sólo para escribir datos.

NOT OUTPUT FILE (El fichero no es de salida)

Se produce cuando se ha intentado grabar datos con PRINT# en un fichero que estaba especificado sólo para leer datos.

OUT OF DATA (No hay más datos)

Se produce cuando se intenta leer con READ más datos de los que hay guardados en instrucciones DATA. Puede solucionarlo añadiendo más datos a la instrucción DATA, incluyendo una instrucción RESTORE o disminuyendo el número de datos a leer en READ.

OUT OF MEMORY (No hay más memoria libre)

Se produce siempre que el programa o sus variables necesitan más memoria de la que el ordenador tiene libre, pero observe dos ejemplos que también presentarían este error.

1. PRINT 1*(1*(1*(1*(1*(1*(1*(1*(1*(1*(1))))))))))

La ejecución de esta orden presenta este mensaje porque el ordenador no puede tener pendientes más de nueve operaciones y en el ejemplo tiene 1Ø.

2. 2ØØØ GOSUB 2ØØØ

   RUN 2ØØØ

Este ejemplo es menos evidente, pero si lo prueba observará que también presenta este mensaje debido a que cada vez que se encuentra un GOSUB el ordenador guarda en memoria la dirección de retorno. Con esta instrucción conseguimos, por tanto, agotar la memoria.
Sucede también si hay muchos bucles anidados o demasiadas subrutinas.

OVERFLOW (Desbordamiento)

Se produce cuando el resultado de un cálculo es mayor que el máximo número permitido, o sea que:

1.7Ø141183 E+38

REDIM'D ARRAY (Matriz redimensionada)

Se produce cuando se dimensiona en el mismo programa una matriz más de una vez, o también cuando no se ha dimensionado una matriz porque no se iba a trabajar con más de 1Ø subíndices y se dimensiona *a posteriori*.

REDO FROM START (Vuelva a empezar)

Se produce cuando el programa se detiene al llegar a una instrucción INPUT en espera de un dato numérico y le suministramos uno alfanumérico; el programa continua normalmente si le suministramos una entrada correcta. La única precaución es que si se trata de un INPUT múltipe la entrada debe repetirse desde el principio.

RETURN WITHOUT GOSUB (RETURN sin GOSUB)

Se produce cuando el ordenador encuentra una instrucción RETURN sin haber encontrado antes una GOSUB. Se puede solucionar insertando una instrucción GOSUB o borrando el comando RETURN. Ejemplo:

```
10 GOSUB 1000
20 PRINT "HOLA"
30 RETURN
```

STRING TOO LONG (Cadena demasiado larga)

Se produce siempre que intentemos que una cadena contenga más de 255 caracteres.

SYNTAX (Sintaxis)

Este es el mensaje de error que quizá con más frecuencia le aparezca, al menos al principio. El C-64 presenta este mensaje cuando no puede reconocer una sentencia porque se ha deletreado mal una palabra clave, por ejemplo, teclear RUM en lugar de RUN o IMPUT por INPUT etc., o bien porque utiliza una coma donde debería ir un punto y coma, por ejemplo:

```
10 INPUT "DAME DATOS", A, B
```

En este caso daría el error porque la primera coma debería ser un ";". También puede surgir porque falte un paréntesis, no se hayan puesto los dos puntos que indican una línea multisentencia, se utilice un número de línea mayor de 63.999, etc.

TOO MANY FILES (Demasiados ficheros)

Se produce al intentar abrir el onceavo fichero, porque no se pueden abrir a la vez más de 10 ficheros, compruébelo con este ejemplo que intenta abrir 11.

```
10 FOR I=1 TO 11
20 OPEN I, 4
30 NEXT I
```

TYPE MISMATCH (Tipo no válido)

Se produce cuando el tipo de dato no es el correcto para el uso que se quiere hacer de él. Por ejemplo, si introducimos un número en una cadena debe ir entre comillas, si no es así aparecerá este mensaje, compruébelo:

```
10 A$=6
20 PRINT A$
```

También aparece cuando intentamos hacer operaciones que no son posibles con cadenas como PRINT "2"*3.

UNDEF'D FUNCTION (Función no definida)

Se produce cuando se hace referencia a una función definida por el usuario, pero que no se ha definido nunca, por ejemplo, PRINT FN (x); también se produce si olvidamos terminar una subrutina con el comando RETURN.

UNDEF'D STATEMENT (No existe esa sentencia)

Se produce siempre que se envía con GOTO, GOSUB o THEN el control a una línea de programa que no existe. También si se intenta ejecutar con RUN a partir de una línea que no existe. Por ejemplo, RUN 55, y no existe ninguna línea 55.

VERIFY (Verificar)

Se produce cuando se intenta verificar un programa del cassette o del disco y éste no coincide con el que actualmente tiene en memoria el ordenador.

Con el siguiente programa podrá conseguir un listado de casi todos los mensajes de error descritos, concretamente de los 29 mensajes que ocupan posiciones de memoria consecutivas a partir de la dirección 41374; el orden en que aparecerán no es alfabético, sino aquel en el que el ordenador los tiene almacenados.

```
 5 REM LISTADO DE ERRORES
10 PRINT "   "
20 DI=41373
30 DI=DI+1
40 PRIN CHR$ (PEEK(DI) AND 127);
50 IF PEEK (DI)< 128 THEN 30
60 PRINT
70 GET A$: IF A$="" THEN 70
80 IF PEEK (DI+1)< 128 AND PEEK
(DI+1)> 31 THEN 30
```

Puede que le resulte extraño el «AND 127» de la línea 40. Lo utilizamos para conseguir poner a cero el bit más significativo del octeto que estamos considerando, y lo hacemos para que la última letra de cada mensaje sea una letra y no un carácter gráfico, como sería si lo suprimimos. ¡Compruébelo!

Es curioso observar que en esta lista no aparecen los dos errores característicos del INPUT:

```
? EXTRA IGNORED
? REDO FROM START
```

Es debido a que el ordenador los tiene almacenados en otras direcciones de memoria, concretamente a partir de la 44284.

Apéndice D: Códigos ASCII

Se muestran a continuación los códigos devueltos por la función ASC que corresponden a cada carácter, así como lo que en cada caso podría observar en la pantalla al imprimir CHR$ (código).

Código	*Efecto*
0	NADA EN PANTALLA
1	NADA EN PANTALLA
2	NADA EN PANTALLA
3	NADA EN PANTALLA
4	NADA EN PANTALLA
5	FIJA EL BLANCO COMO COLOR DE LA TINTA
6	NADA EN PANTALLA
7	NADA EN PANTALLA
8	ANULA EL PASO A MINUSCULAS PRODUCIDO AL PULSAR SIMULTANEAMENTE LAS TECLAS SHIFT 4 y
9	NEUTRALIZA A CHR$(8)
10	NADA EN PANTALLA
11	NADA EN PANTALLA
12	NADA EN PANTALLA
13	MISMO EFECTO QUE PULSAR [RETURN]
14	MODALIDAD MINUSCULAS
15	NADA EN PANTALLA
16	NADA EN PANTALLA
17	IGUAL QUE [CRSR ⇩]
18	ACTIVA [RVS ON]
19	IGUAL QUE [CLR HOME]
20	IGUAL QUE [INST DEL]

Código	*Efecto*
21	NADA EN PANTALLA
22	NADA EN PANTALLA
23	NADA EN PANTALLA
24	NADA EN PANTALLA
25	NADA EN PANTALLA
26	NADA EN PANTALLA
27	NADA EN PANTALLA
28	FIJA EL ROJO COMO COLOR DE LA TINTA
29	IGUAL QUE [CRSR ⇨]
30	FIJA EL VERDE COMO COLOR DE LA TINTA
31	FIJA EL AZUL COMO COLOR DE LA TINTA
32	PRODUCE UN ESPACIO EN BLANCO
33	——— !
34	——— "
35	——— #
36	——— $
37	——— %
38	——— &
39	——— '
40	——— (
41	———)
42	——— *
43	——— +
44	——— ,
45	——— -
46	——— .
47	——— /
48	——— 0
49	——— 1
50	——— 2
51	——— 3
52	——— 4
53	——— 5
54	——— 6
55	——— 7
56	——— 8
57	——— 9
58	——— :
59	——— ;

Código	Efecto
60	<
61	=
62	>
63	?
64	@
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z
91	[
92	£
93	]
94	↑
95	←
96	
97	
98	
99	
100	
101	
102	
103	
104	
105	
106	
107	
108	
109	
110	
111	
112	
113	
114	
115	
116	
117	
118	
119	
120	
121	
122	
123	
124	
125	
126	
127	
128	NADA EN PANTALLA
129	NADA EN PANTALLA
130	NADA EN PANTALLA
131	NADA EN PANTALLA
132	NADA EN PANTALLA
133	CORRESPONDE A LA TECLA f1
134	CORRESPONDE A LA TECLA f2
135	CORRESPONDE A LA TECLA f3
136	CORRESPONDE A LA TECLA f4
137	CORRESPONDE A LA TECLA f5
138	CORRESPONDE A LA TECLA f6
139	CORRESPONDE A LA TECLA f7
140	CORRESPONDE A LA TECLA f8

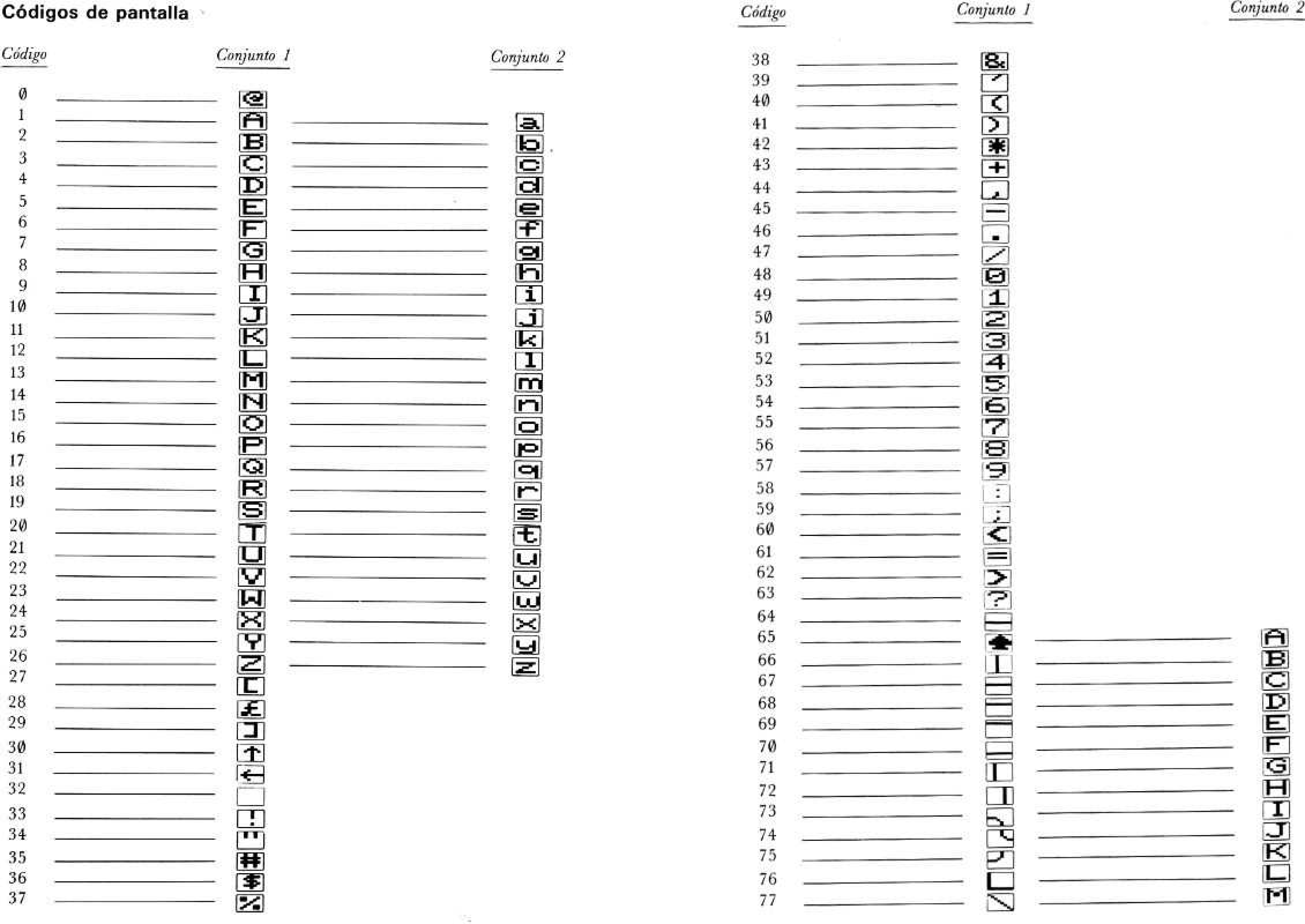
Código	*Efecto*
141	MISMO EFECTO QUE [SHIFT] + [RETURN]
142	MODALIDAD MAYUSCULAS
143	NADA EN PANTALLA
144	FIJA EL NEGRO COMO COLOR DE LA TINTA
145	IGUAL QUE [⇧ CRSR]
146	NEUTRALIZA A CHR$ (18)
147	IGUAL QUE [CLR HOME]
148	IGUAL QUE [INST DEL]
149	NADA EN PANTALLA
150	NADA EN PANTALLA
151	NADA EN PANTALLA
152	NADA EN PANTALLA
153	NADA EN PANTALLA
154	NADA EN PANTALLA
155	FIJA EL PURPURA COMO COLOR DE LA TINTA
156	IGUAL QUE [⇦ CRSR]
157	FIJA EL AMARILLO COMO COLOR DE LA TINTA
158	FIJA EL AZUL CLARO COMO COLOR DE LA TINTA
159	MISMO EFECTO QUE LA TECLA SPACE
160	
161	
162	
163	
164	
165	
166	
167	
168	
169	
170	
171	
172	
173	
174	
175	
176	
177	

Código	*Efecto*
178	
179	
180	
181	
182	
183	
184	
185	
186	
187	
188	
189	
190	
191	
192	
193	
194	
195	
196	
197	
198	
199	
200	
201	
202	
203	
204	
205	
206	

Código	*Efecto*
207	
208	
209	
210	
211	
212	
213	
214	
215	
216	
217	
218	
219	
220	
221	
222	
223	
224	
225	
226	
227	
228	
229	
230	
231	
232	
233	
234	

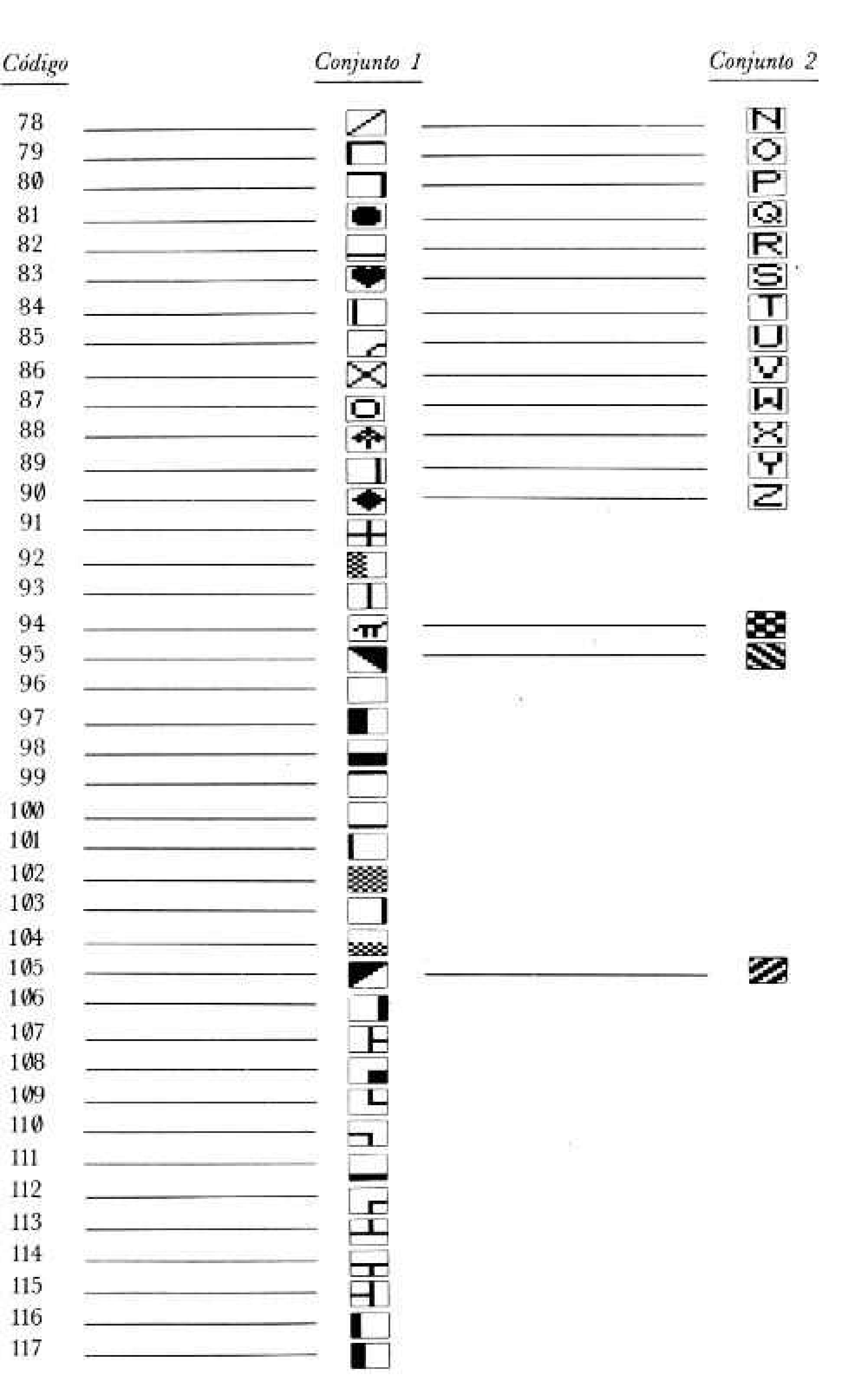
Compruebe que:

Los códigos del 192 al 223 son los mismos que del 96 al 127. Del 224 al 254 son los mismos que del 160 al 190 y que el código 255 es el mismo que el 126.



Códigos de pantalla

Código	*Conjunto 1*	*Conjunto 2*
0	@	
1	A	a
2	B	b
3	C	c
4	D	d
5	E	e
6	F	f
7	G	g
8	H	h
9	I	i
10	J	j
11	K	k
12	L	l
13	M	m
14	N	n
15	O	o
16	P	p
17	Q	q
18	R	r
19	S	s
20	T	t
21	U	u
22	V	v
23	W	w
24	X	x
25	Y	y
26	Z	z
27	[	
28	£	
29	]	
30	↑	
31	←	
32		
33	!	
34	"	
35	#	
36	$	
37	%	
38	&	
39	'	
40	(	
41	)	
42	*	
43	+	
44	,	
45	-	
46	.	
47	/	
48	0	
49	1	
50	2	
51	3	
52	4	
53	5	
54	6	
55	7	
56	8	
57	9	
58	:	
59	;	
60	<	
61	=	
62	>	
63	?	
64	─	
65	♠	A
66	│	B
67	─	C
68	─	D
69	─	E
70	─	F
71	│	G
72	│	H
73	╮	I
74	╰	J
75	╯	K
76	└	L
77	╲	M



Código	Conjunto 1	Conjunto 2
78		N
79		O
80		P
81		Q
82		R
83		S
84		T
85		U
86		V
87		W
88		X
89		Y
90		Z
91		
92		
93		
94		
95		
96		
97		
98		
99		
100		
101		
102		
103		
104		
105		
106		
107		
108		
109		
110		
111		
112		
113		
114		
115		
116		
117		

Código	Conjunto 1	Conjunto 2
118		
119		
120		
121		
122		
123		
124		
125		
126		
127		

Epílogo

¿Y ahora qué? Bien, en primer lugar, «felicidades». El primer paso ya está dado. Sin embargo, es obvio que hay mucho más que aprender sobre el Commodore: detalles sobre otras particularidades del BASIC, otros puntos sobre los periféricos existentes y los que aparecerán en el futuro, ¿cómo aplicar todo esto a mi ámbito?

Ante todo practique. Las técnicas de programación, los detalles sobre los errores, los nuevos conocimientos se adquieren practicando. Siéntase libre, experimente. Ensaye sin temor sus nuevas ideas. El arte de la programación se basa en el refinamiento sucesivo. Al principio, las ideas surgen en bruto en la cabeza, después se nos ocurre que podríamos haber simplificado mucho las cosas si hubiéramos hecho... Poco a poco, cuando vamos resolviendo el enésimo problema, las ideas surgen ya más refinadas y los pasos posteriores se van simplificando.

Es aconsejable, al principio, seguir un libro de ejercicios o un libro cuyo contenido verse sobre nuestro tema de interés. En ellos encontrará muchos trucos o soluciones que poco a poco irán formando parte de su biblioteca personal. Cada vez que encuentre algo nuevo, apúntelo; así, después sabrá dónde está, por si necesita volver a utilizarlo.

Y cuando crea que ya lo ha aprendido todo y que sus programas son inmejorables, recuerde al corolario informático de las leyes de MURPHY:

«NO EXISTEN PROGRAMAS PERFECTOS, SOLO EXISTEN PROGRAMAS CON ERRORES AUN NO DETECTADOS.»

Esta obra está planteada para ayudar al lector *desde el principio* a manejar y sacar el máximo partido a su Commodore. El principal objetivo es que domine el lenguaje BASIC con el que su ordenador viene equipado introduciéndolo de manera gradual en el mundo de la informática.

Las primeras páginas del libro se destinan a enseñar al usuario el funcionamiento de su máquina y a familiarizarlo con el teclado, herramienta indispensable para comunicarse con el ordenador.

Cada uno de los programas que se presentan después aportarán al lector ideas nuevas que le permitirán aplicar los conceptos adquiridos a la creación de *sus propios programas*.

